Optimization of Test Cases using Soft Computing Techniques: A Critical Review

Manoj Kumar[#1] Arun Sharma[2] Rajesh Kumar[3]
[1]Department of Computer Application, (Galgotias University), Greater. Noida,(U.P.) - India,
[2]Department of Computer Sc., KIET, Ghaziabad,U.P. –India,
[3]School of Mathematics & Computer Application, (Thapar University) Patiala- India,
m_pachariya1@yahoo.com, http://www.galgotiasuniversity.edu.in

Abstract: - Software testing is the key technology for evaluating the fault detecting capability quantitatively. Software testing is very labor-intensive and expensive process. It is a core activity in quality assurance. Test cases minimization, selection, prioritization forms common thread of optimization. Test case optimization is a multi-objective optimization, peculiar nature and NP-Complete problem. However, by applying appropriate test case optimization techniques, these efforts can be reduced considerably. Moreover, by using the multi-objective optimization of test cases with test data adequacy criteria and automation of testing process will help in improving the overall quality of the software. Present paper gives the insight into existing single objective test cases optimization techniques such as Genetic Algorithms, Ant Colony Optimization, Hybrid Genetic, Intelligent Search Agent Techniques, Particle Swan Optimization, Graph based Intelligent Techniques, Hybridization of Soft Computing techniques devised by various researchers or practionners by using single parameter like number of defect detecting capability, cost, efforts, coveragebility of requirement/ code and quality of the results. In addition to this, it highlights some research issues relating to above.

Key-Words: - Multi-Objective Optimization, Soft Computing Techniques, Test Cases, Test Data Adequacy Criteria.

1 Introduction

Software testing plays a vital role in quality software development. Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user. Although software testing is a very labor-intensive and itself an expensive activity, yet launching of software without proper testing may lead to cost potentially much higher than that of testing, specially in systems where human safety is involved[1,2]. If the process of testing could be automated, significant reductions in the cost of software development can be achieved. It depends on the quality/fitness and number of test cases exercised. The solution is to choose the most important and effective test cases and removing the redundant and unnecessary ones, which in turn leads to test case optimization[3,4]. A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.

Because of the lack of known strategies, decisions like these are made on the basis of the experience, intuitive assessments and heuristic rules. Existing literature review has identified some key problems in software testing related to test cases selection, test cases prioritization, test cases minimization, corresponding adequacy criteria, and analysis of impact of test cases on software quality. Decision makers have to answer the following questions with economic criteria: What test cases shall tester use to exercise the program?, How to select the test cases with maximum coverage ability?, When and how to determine whether testing has been conducted adequately?, When to stop testing and whether to continue the testing?, When to stop optimization and whether to continue the optimization?, How to determine that whether generate optimized test cases or optimize the randomly generated test cases, which one is the better approach?, How to determine the quality of software from test cases?, When the quality of software should be evaluated by means of directed testing where software is frequently changing?, What will be the probability of test case failure? and others[5,6].

Software test adequacy criteria are the rules to determine whether a software system has been adequately tested, which points out the central problem of software testing i.e. “what is a test data adequacy criterion?”. Number of test data adequacy criteria has been proposed and investigated in the

literature like control flow-based test adequacy criteria, data flow based adequacy criteria, fault-based adequacy criteria, and error-based criteria. Control flow-based adequacy criteria includes statement coverage, branch coverage, path coverage, Length-i path coverage, loop coverage, relational operator coverage, table coverage(whether each entry in a particular array has been referenced), cyclomatic number criterion. Data-flow based adequacy criteria includes all definitions criterion, all uses criterion. Fault-based adequacy criteria include error seeding and mutant coverage or mutant killing score. Each criterion has its own strength and weakness. A central question in the study of test adequacy criteria is that how they relate to fault detecting ability [7,8,9].

The effectiveness of this verification and validation process depends upon the number of errors found and rectified before releasing the system. This, in turn, depends upon the fitness of test cases generated. A test case is an input to the program under test. It is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. It is the mechanism for determining whether a software program or system has passed or failed. Data generation for software testing is the process of identifying program input, which satisfy testing criterion. There are two different approaches taken by test data generators, namely, path oriented and goal oriented approach [8,10]. Usually, the number of test cases required to develop error-free software, will be very high. Since, exhaustive testing is not possible, the generated test cases should be optimal and also cover the entire software and reveal as many errors as possible [11,12]. Automatic generation of optimized test cases is one of difficult points of this technology [12]. A test suite, less commonly known as a validation suite, is a collection of test cases that are intended to be used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests. Occasionally, test suites are used to group similar test cases together for some specific functionality of the system. Test suite may contain some redundant, irrelevant test cases. Since, testing is very expensive process, unnecessary execution of redundant, irrelevant test cases will increase unnecessary burden of cost. So, test cases optimization is necessary [13]. Test suite minimization is to find minimal cardinality sub set of test suite, which exercises same set of requirements as exercised by un-minimized test suite. Test suite minimization is minimal set cover problem, which uses greedy approximation approach to solve it. So, Test suite minimization is NP-complete problem [14,15,16]. Peculiar nature problem are those problems which requires curious mix of data and knowledge driven approach to solve it. Test cases optimization is a search space problem, which requires hybridization of data driven and knowledge driven approach to find near optimal solution of the problem. Hence, Test cases optimization is also peculiar nature problem[14,17].

An optimization problem is the problem of finding the best solution from all feasible solutions. Multi-Objective optimization (MO) also known as multi-criteria or multi-attribute optimization, is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. The objective of MO optimization is to find the set of acceptable solutions and present them to the decision maker/Tester to take decision. If a multi-objective problem is well formed, there should not be a single solution that simultaneously minimizes each objective to its fullest. In each case an objective must have reached a point such that, when attempting to optimize the objective further, other objectives suffer as a result. Finding such a solution, and quantifying how much better this solution is compared to many other such solutions, is the goal when setting up and solving a multi-objective optimization problem. Test cases optimization is the problem of finding the best sub set of test cases from a pool of test case to be audited. It will meet all the objectives of testing concurrently. Test suite should be designed in such way that it will achieve maximum of code coverage, maximum requirements coverage, high fault detecting capability, maximum mutant killing score. Test suite should also contain minimum number of test cases with minimum execution efforts and cost. Test suite should be constructed in minimum efforts, time and cost. So, Test cases optimization is a multi-objective optimization problem.

Test cases generated by combining variable input values and variable sequencing of input values result in too many possible combinations to test. Due to time limitations on testing, all possible combinations of input values and sequencing of input results cannot be executed [18,19]. As such, test cases optimization techniques such as test case selection, test suite minimization and prioritization are considered. Thus, it becomes essential to optimize the test cases in order to cover maximum

faults in minimum time. The objective of test case optimization is to reduce the number of test cases and to improve the fitness/effectiveness of test cases. Since, test cases optimization problem is a multi-objective optimization problem, It cannot be solved in combinatorial time and hence it is a NP-hard problem. Solution to these type problems cannot be obtained by direct search. It requires heuristic searching techniques [15,20]. Though, there are several objectives of test case optimization like maximum number of defect detecting capability, minimum test design efforts/cost, minimum execution cost, maximum coveragebility of client requirements and codes, maximum mutant killing score and so forth. Therefore optimization of test cases should be treated as multi-objective optimization problem. However most of test cases optimization approaches found in the literature are single objective. Single objective formulation of test cases optimization problem is not justified and not meeting the objectives of testing. Some objectives of test cases optimization are conflicting in nature, coveragebility of one objective will suffer other objective like cost, quality and quantity while considering all objectives concurrently. It is not appropriate to estimate fitness of test cases or optimize the test cases just on the basis of only a single parameter/objective as it is a multi-faceted concept. So, Multi-faceted concept for test cases optimization will help tester /decision maker to take decision for selecting/filtering prioritizing the test cases with highest adequacy. It will provide testers/decision maker the fitness/coveragebility scores of test cases for opting best test case from various alternate test cases. It will surely reduce the cost & efforts of software testing and improve the quality of test cases and reduce the number of test cases also. Hence, the study of existing literature concluded that the test cases optimization is a multi-objective optimization, peculiar nature and NP-Complete problem. Present paper discusses these several issues and conducts a critical survey for various proposals made by researchers for test case optimization by using soft computing techniques.

2 Test Cases Optimization

Software testing consists of three main activities: selecting tests inputs, running the inputs on the software under test, and evaluating the correctness of the outputs. The first and third of these activities are labour-intensive and error prone. Software testing and retesting occurs continuously during the software development lifecycle to detect errors as early as possible and to ensure that changes to existing software do not break the software. Test suites once developed are reused and updated frequently as the software evolves. As a result, some test cases in the test suite may become obsolete, redundant as the software is modified. Due to the resource and time constraints for re-executing large test suites, it is mandatory to optimize available test suites by using test cases prioritization, test case filteration, test case selection and test suite minimization [21].

Test case prioritization techniques try to find an ordering/ranking of test cases so that some test case adequacy can be maximized as early as possible. Test case prioritization and filteration depend on quality of initial population of test cases. Selection and prioritization of test cases are the two important solutions to the problem of test case optimization. Test case filteration and prioritization are closely related. In fact, test cases can be filtered by selecting the first N ordered test cases. Therefore, any test case prioritization algorithm can be used as a test case selection algorithm. Naturally, it is desirable to select those test cases that are most likely to reveal defects in the program under test [22,23]. Test suite minimization is a selection of smallest subset the test cases from a pool of test cases to be audited for a program. It covers as many program elements as the entire pool does. Test suite reduction seeks to reduce the number of test cases in a test suite while retaining a high percentage of the original suite's fault detection effectiveness. Test suite minimization techniques seek to reduce the effort required for regression testing by selecting an appropriate subset of test suites. The test suite minimization problem is a special case of the traditional set-cover problem, which is NP-hard [24,25]. One major difference between test suite minimization and test case selection is that test case selection chooses a temporary subset of test cases, whereas test suite minimization reduces the test suite permanently based on some external criterion such as structural coverage. When testing a program, software testers have to define the testing objectives first. A test suite is then constructed to satisfy the all objectives of testing. It is generally agreed that a test suite must achieve maximum coveragebility of all objectives of testing [26,27]. Usually, the constructed test suite may contain redundant test cases. A test case in a test suite is said to be redundant if the same testing objective can still be satisfied by other test cases of the test suite. Since the execution of test cases and evaluation their results are very expensive, it is of paramount importance to remove redundant test cases within a test suite. However, removal of all redundant test

cases is practically infeasible because the problem is NP-complete[28]. However, a weakness of test suite reduction is that the removal of some test cases from the test suite may potentially reduce the fault detecting capability of the test suite too. To be worthwhile, the sum of the cost of test cases filteration, execution and audit of selected test cases should be less than the cost of that of all of the test cases of the original pool [23, 25]. The goal of test cases filteration is to chunk/filter out irrelevant, redundant and less fit test cases from the test suite. Test cases filteration is to hunk out subset of closely related test cases. So that a large portion of the defects would be found as if the whole test suite was to be used. It is often desirable to filter a pool of test cases for a program in order to identify a subset that will actually be executed and audited at a particular time. When it is uncertain that how many test cases can be run and audited, it is advantageous to order or rank the test cases as per priorities so that the tester will select the test cases as per their rank or order, which permit tester to start quick, early fixing the most of the defects [29,30]. Study of existing literature has identified several techniques for test cases optimization. Some of them are described below:

Xiao Qu et al. [9] proposed a combinatorial interaction techniques (CIT) for regression testing for test case generation and prioritization. It is an effective regression testing techniques to select and order (or prioritize) test cases between successive releases of a program. However, selection and prioritization are dependent on the quality of the initial test suite. An effective and cost efficient test generation technique is combinatorial interaction testing (CIT), which systematically samples all t-way combinations of input parameters. They examined several CIT prioritization techniques and compare them with a re-generation/prioritization technique. They concluded from results that CIT performs well in finding seeded faults when compared with an exhaustive test set. It lacks the prioritization of test cases on several criteria concurrently. Kim et al. [30,31] proposed test cases prioritization techniques for regression testing using test historical data to reduce the cost of regression testing. Under certain conditions, some can even guarantee that the selected test cases perform no worse than the original test suite. They prioritized the test cases and exercised only those that fit within existing constraints. They pointed out that existing prioritization techniques are memory less, implicitly, local choices. Instead, they proposed a new technique for prioritization based on historical execution data and conducted an experiment to assess its effects on the long run performance of resource constrained regression testing. It lacks its validity and generalization. Varun et al. [32] proposed a new approach for test case prioritization techniques using fault severity based on requirement prioritization. Aim is to find the severity of faults early in the testing process and hence to improve the quality of the software according to customer point of view, to reduce the cost of regression technique and to increase the effectiveness of testing process. They formulated testing objective to increase rate of fault detection early in the testing process and others formulated the objectives based on code coverage and focused for finding the maximum no of faults rather than severity of faults. They concluded that prioritization approach frequently yields faults with high severity from their experiment results.

Leon et al. [21,33] studied an empirical comparison of four different techniques for filtering large test suites(test suite minimization, prioritization) by using additional coverage, cluster filtering, one-per-cluster sampling, and failure pursuit sampling. The first two techniques are based on selecting subsets that maximize code coverage as quickly as possible, while the latter two are based on analyzing the distribution of the tests' execution profiles. These techniques were compared with data sets obtained from three large subject programs: the GCC, Jikes, and javac compilers. Some simple combinations of these techniques were evaluated for use in test case prioritization, and found that these two kinds of techniques are complementary in the sense of finding different defects.. The results also indicate that these techniques can create more efficient prioritizations than those generated using prioritization by additional coverage. They concluded from results obtained that distribution-based techniques can be as efficient or more efficient for revealing defects than coverage-based techniques. Harrold et al. [23] proposed and experimented a technique to select a representative set of test cases from a test suite that provides the same coverage as the entire test suite. This selection is performed by identifying and then eliminating, the redundant and obsolete test cases in the test suite. It is not dependent on any particular test selection criterion and can be used as long as the association between requirements and test cases can be made. They developed a program and applied the proposed techniques on the program to identify and remove unnecessary test cases. They found a significant reduction in fault detection capability of test suite from experiment when they exercised minimized test suite on a simple program.

Fischer et al. [34] proposed a formal method for selection of test cases for regression testing. Their approach required both control flow and data-flow analysis to determine which test case should be selected. Miller et al. [35] pointed out the need for the identification of program units where all changes can be localized, while addressing the issues involved in the automation of software development processes. Benedusi et al. [36] proposed a test selection technique based on path change analysis, and constructed a test table, containing test cases and associated paths. This reduction of number of rows is accomplished based on the coverage of input conditions and paths. The test cases are selected on the basis of minimum number of rows in the test table. Weiser et al. [37] proposed dynamic slicing technique for test case selection. These slicing techniques were applied in program debugging. The objective of their research was to extract a small portion of the code that possibly contains the more faults. Subsequently, these techniques were applied to Regression Test cases Selection (RTS). Agrawal and Horgan [38,39] have proposed several dynamic slicing algorithms, also described dynamic slicing in the presence of unconstrained pointers for regression test case selection. Ferrante et al. [40] proposed an optimization approach using Program Dependency Graph (PDG) that makes explicit both the data and control dependence for each operation in a program. Data dependences have been used to represent only the relevant data flow relationships of a program. Control dependences are introduced to analogously represent only the essential control flow relationships of a program. Control dependences are derived from the usual control flow graph. The PDG allows transformations such as vectorization. Subsequently PDG was used for selection of regression test cases.

Frankl et al. [5] proposed an analytical approach for fault detecting ability of testing methods. They examined several relations between software testing criteria, each induced by a relation between the corresponding multi sets of sub domains. They explored whether for each relation R and each pair of criteria, C1 and CZ, R(C1, Cz) and investigated that C1 is better at detecting faults than CZ according to various probabilistic measures of fault-detecting ability. They concluded that the fact that C1 subsumes CZ does not guarantee that C1 is better at detecting faults. Wong et al. [11,27] have reported several studies aimed at evaluating the fault-detection effectiveness of test cases prioritization and test suite minimization using all blocks, decisions, and the other criteria while retaining one or more control-flow and data-flow-based coverage metrics. Size and code coverage are important attributes of a set of tests. They addressed the issue "What is the impact of reducing the size of test suite on fault detecting capability, while keeping coverage constant?". They found little to no loss in the fault-detection effectiveness, when test cases that do not reduce overall block coverage, are removed from a test suite. Tallam and Gupta [12] proposed a delayed greedy minimization algorithm using lattices and dominators for test cases minimization. It improves upon the prior heuristics by iteratively exploiting the implications among the test cases and the implications among the coverage requirements, leveraged only independently from each other. They experimented comparable time performance and concluded that proposed techniques consistently produced same size or smaller size test suites than prior heuristics. They also admitted that the test suite minimization problem is NP complete. Harrold et al. [25] proposed a methodology to control the size of a regression-test suite. Test-suite reduction based on code coverage is one of the several criteria proposed by authors for the reduction in the size of a test suite. Lei et al. [41] proposed a framework for minimization of randomized unit test cases and gave empirical evidence that randomized generation of unit test cases using sequences of method calls has high coveragebility and more effective. The randomized test generator allows the automatic production of a high volume of varied test input, and the test oracle allows the output to be checked automatically. The goal is not to build a fixed test suite, but rather to keep drawing test cases randomly from a large test case space until either the software under test fails or a stopping condition is reached. Authors showed that test case minimization algorithm significantly reduces the length of these sequences. They studied the resulting benefits qualitatively and quantitatively, via a case study on open-source data structures and an experiment on lab-built data structures. They concluded that randomized unit test cases can achieve high coverage and is effective at forcing failures, but tends to generate long failing test cases. They used algorithm proposed by Zeller and Hildebrandt [13] for Simplifying and isolating failure-inducing input . It can significantly reduce the length of these failing test cases, making them more valuable for the debugging process. This in turn increases the practical applicability of randomized unit testing. It lacks the implementation a JUnit-style or Jartege- style framework for automation of randomized testing and test case

minimization of unit testing. This technique has been known for years in specific domains, like functional program testing and compiler testing . Agrawal et al. [42,43] used the notion of dominators, superblocks and mega blocks to derive coverage implications among the basic blocks to reduce test suites such that the coverage of statements and branches in the reduced suite implies the coverage of the rest. Similarly, Marre and Bertolino [44] used a notion of entities subsumption to determine a reduce set of coverage entities such that coverage of the reduced set implies the coverage of un-reduced set. Sampath et al. [45] have presented a concept analysis based algorithm (SMSP) for reducing a test suite for web applications. They considered the URLs used in a web session as the attributes and each web session as a test case. One test case from each of the strongest concept in the concept lattice is selected to generate a reduced test suite to cover all the URLs covered by the unreduced suite. As shown in recent report of Sprenkle et al. [28], the reduced suites produced by their approach are in generally larger than those produced by applying the classical greedy algorithm and the HGS algorithm for reduce a set of web user sessions. Sampath [45] also experimented and concluded that Delayed-Greedy algorithm always produced equal or smaller test suites than classical greedy algorithm, the HGS algorithm [25] and the SMSP [45] algorithm. Rothermel et al. [46] made an empirical study for analysing the impact of size reduction test suite/set on fault detecting capability of test case. They used HGS Algorithm for reducing/minimizing test suite. They concluded that size reduction of test suite will certainly reduce the fault detecting capability of test suite, which is contradiction of Wong studies[11, 27]. Horgan and London [47] implemented a tool a for data flow coverage testing in C programming language, called ATAC. It was implemented to construct the optimized test suite.

The works in [11,27,46] study the effects of test suite minimization on the fault detection capabilities of the reduced test suites. In [46], the HGS algorithm is used for minimization of test suites selected from the test suite pools. Thus, the quality of the test suites selected from these test pools is high as they contain test cases to cover a wide range of requirements. Therefore, in the experimental studies reported in [46], a significant loss in the fault detection capability of the minimized suites was observed. In contrast, the experimental studies in [11,27] used ATAC [47] system to compute optimally minimized test suites from the randomly generated test suites. It can be concluded that minimization techniques can reduce the test suite size to a great extent with no loss in fault detection capabilities of test suites. Although, these two studies seem to be contradictory, we believe that the quality of the initial test suites used in detecting the faults and experimented with software under test is the only fundamental reason for the clashing conclusions obtained in these studies [11,27,46].

Jones and Harrold [48] have recently presented some heuristics to minimize test suites specifically tailored for the Modified Condition/Decision Coverage (MC/DC) criterion. Authors proposed a heuristics techniques for reducing a test suite with respect to set of requirements which could be derived from any coverage criterion or a combination of different criteria. The context table which contains the information about the set of requirements covered by each test case in the test suite is the only input to algorithm. Authors selected the test suite on the basis of maximum test suite score the outcome of algorithm. Graves et al. [49] examined the costs and benefits of several regression test selection techniques, including test suite minimization (greedy coverage maximization), a dataflow technique, a safe technique, and random selection. In separate studies, Elbaum et al. [22,50] and Rothermel et al. [51,52] compared several test case prioritization techniques, including ones based on code coverage, estimated fault proneness, and other factors using historical execution data.

3 Soft Computing Techniques for Test case Optimization

Software testing plays a crucial role in high quality software development. It uses the application of artificial intelligence techniques, that in turn helps in identifying optimized test cases which will improve quality of testing, reduce the total time and cost needed in the testing process. The paradigm of soft computing or computational intelligence refers to the seamless integration of different, seemingly unrelated, intelligent technologies such as fuzzy logic, NNs, GAs, ML (CBR and decision trees subsumed), rough set theory and probabilistic reasoning in various permutations and combinations to exploit their strengths in the area of software testing. Soft computing is an emerging collection of methodologies, which aim to exploit tolerance for imprecision, uncertainty, and partial truth to achieve robustness, tractability and total low cost. Soft computing is a term applied to a field within computer science which is characterized by the use

of inexact solutions to computationally-hard tasks such as the solution of NP-hard problems, for which an exact solution cannot be derived in polynomial time [53,54]. Soft Computing techniques are also providing better solution of peculiar nature problems, which is curious mix of data and knowledge driven problem. Problem of generating a minimum test suite or test cases optimization is also NP-complete and peculiar nature problem. The exemplar of computational intelligence techniques in the area of software testing is less explored but some researchers have explored soft computing techniques in this area are as follows:

Mala et al. [15] proposed Hybrid Genetic Algorithm (HGA) based approach for improving the software quality by optimization of test cases. HGA approach combines Genetic Algorithm (GA) and Local Search (LS) techniques to reduce the number of test cases by improving quality of test cases during test case generation process. They compared the proposed approach with Genetic Algorithm (GA), Bacteriologic Algorithm (BA) and concluded that HGA is best among them. Berndt et al. [55] proposed a breeding techniques for optimization of software test cases with the help of Genetic Algorithms. They used an evolving fitness function. They also proposed a framework that distinguishes between absolute and relative fitness functions. It is used to organize past research and characterize this project's reliance on a relative or changing fitness function. In particular, the genetic algorithm includes a fossil record that records past organisms, allowing any current fitness calculations to be influenced by past generations. Three factors are developed for the fitness function: novelty, proximity, and severity. They developed several techniques for fossil record visualization are developed and used to analyze different fitness function weights and resulting search behaviors. Debasis et al. [56] used genetic algorithm to optimize the test cases, generated graph using the category-partition and test harness patterns. They investigated an approach for measuring effectiveness of test cases, The optimal test suites are devised by the method of sampling statistics. Prabahar et al. [57] used Hybrid Genetic Algorithm (HGA) to optimize the test cases, and compared the simple genetic algorithm with HGA for optimization of test cases. They concluded that HGA is better than simple GA. Baudry et al. [58] explored several complementary computational intelligence techniques for testing of .Net component. They used new artificial Intelligent (AI) algorithm to estimate the defect revealing power of test cases, and automatically improving test cases efficiency. They also explored GA to estimate the defect revealing power of test cases, and automatically improving test cases efficiency. They also conducted comparative analysis of GA and BGA (Bacteriological GA) and concluded that BGA is better than GA. Panda et al. [59] proposed graph theory based GA approach for optimization of test cases. They used the predictive modelling based approach for the test cases generation. It uses directed graph of all immediate state of system for expected behaviour of system. They used genetic algorithm for network testing or system testing. The process of figuring out the multiple test cases leads to complications and there are chances to miss out some of the test cases in this process.

Dorigo et al. [60,61] proposed the ACO algorithms, based on pheromone trails used by the ants, which mark out food sources. ACO is probabilistic techniques that can be applied to generate solutions for combinatorial optimizations problems. The artificial ants in the algorithm represent the construction procedures for the stochastic solutions. There are two major problems commonly associated with state-based software testing: (1) some of the generated test cases are infeasible; (2) inevitably many redundant test cases have to be generated in order to achieve the proper testing coverage required by test adequacy criteria [60,61,62]. Though ACO is next generation technique for optimization problems but it is not providing good solutions of problems like multiple objectives optimization, Dynamic Optimization Problems, the Stochastic Optimization Problems, continuous optimization and Parallel Implementations of the constraints. Dorigo et al. [61] proposed search strategy for positive feedback (autocatalytic) process prompts all ants to choose the shorter path. Leading ant is moving towards destination, suddenly an obstacle appears in the path or the path is cut off. Leading ant have to decide whether to turn right or left, The choice is influenced by the intensity of the pheromone trails left by preceding ants. A higher level of pheromone on the right path gives an ant a stronger stimulus and thus a higher probability to turn right. If leading ant followed wrong path due to lack in sensing capability or vaporization of pheromone, the follower ants will also follow wrong path. So, there is a scope of research to identify approach/ technique for predicting impediments of the path. Huaizhong et al. [62] proposed an ant colony optimization approach for automatic generation of test cases sequence/rank in test suite using UML for state-based software testing. Test sequences in a test suite can be automatically generated to achieve

required test coverage. They used an developed algorithm, which is uses the concept that a group of ants can effectively explore the UML State chart diagrams and automatically generate test sequences to achieve the test adequacy requirement. They pointed out the advantages of proposed approach as follows : (1) the UML State chart diagrams exported by UML tools are directly used to generate test sequences; (2) the whole generation process is fully automated; (3) redundant exploration of the State chart diagrams is avoided due to the use of ants, resulting in efficient generation of test sequences. Singh et al. [63] proposed an ant colony optimization based approach for selection and prioritization of test cases. They compared ACO with other techniques using Average Percentage of Faults Detection (APFD) as a parameter and concluded that ACO is providing better results than others. It lacks automation of the techniques and application on large, complex software. They considered single parameter/objective for optimization of test case but it is a multi-objective optimization problem.

Mala et al. [64] proposed non-pheromone based approach for software test suite minimization by using Artificial Bee Colony (ABC) optimization approach, based on intelligent behavior of biological bees. They explored to find near global optimal solution. They compared it with GA, and concluded that ABC approach takes less iteration to complete the task, and found it more scalable. Mohan.V. et al. [8] proposed an Intelligent Search Agent (ISA) technique for optimizing test sequences by using graph, it satisfy the fitness criteria of test sequence. they compared ISA and ACO techniques, and concluded that ISA is taking less time and cost in generating optimal test sequences. Shihab et al. [65] proposed a framework for Intelligent Meaningful Test Data Generation Model (IMTDG). They improved the test data generation, by providing the flexibility to user to insert or select the test data list. Crina Grosan et al. [66] used high dimensional functions for global optimization of test cases. High dimensional function is a certain class of functions, have the property, the partial derivatives have the same equation with respect to all variables. They used the optimum value (minimum or maximum) takes place at a point where all the variables have the same value, to minimize the computational burden due to the fact that the search has to be performed only with respect to one variable for test cases optimization. Ibrahim et al. [67] proposed two techniques for intelligent selection of test-cases that achieves the best coverage using the minimum number of computing cycles, is crucial for microprocessor design. First, it addresses the generalization of covering problems to partial covering. Second, it finds a good set of test cases that fulfils the target coverage under different scenarios, while taking into considerations operations priority, and computing cycles required by each test case.

Wenyan et al. [68] proposed a technique for generation and reduction of test cases using covering rough sets. Authors have proposed a high-dependable method for the generation and reduction of software test cases, which based on software operational profile and covering rough set. Authors tried to improve the efficiency of software reliability testing by using fewer test cases of covering all operational profiles. This method makes up for the shortcoming of Musa method that generates test cases with high repeatability and relatively low efficiency, and consequently improves efficiency of test cases to some extent. Authors also provided one new train of thought for high dependability software testing. Shin et al. [26] introduced the concept of Pareto efficiency for test case selection. The Pareto efficient approach takes multiple objectives such as code coverage, past fault-detection history and execution cost, and constructs a group of non-dominating, equivalently optimal test case subsets. Authors also described the potential benefits of Pareto efficient multi-objective test case selection and illustrated with empirical studies of two & three objective formulations. It lacks applicability a wider range of software artifacts with different meta-heuristic multi-objective optimization techniques. Junmin et al. [69] designed some artificial immune operators for generating optimized test cases. Artificial immune operators play an important role to support the test case generation method by utilizing optimization ability of artificial immune algorithm these originally random generated test cases are continuously optimized till final/finding test case corresponding to the target path by artificial immune analysis. Authors concluded from experimental results manifest that the proposed immune operator designing algorithm is valid and efficiently generate target path test case. Swain et al. [70] proposed a comprehensive test case generation approach with the help of UML models. Authors constructed Use Case Dependency Graph (UDG) with the help of Use Case diagram and Concurrent Control Flow Graph (CCFG). They generated test cases for integration and system testing. They also compared proposed approach with existing approaches on single parameter code coverage and concluded that proposed approach is

the best. However they have not included other important parameters for efficiency of test cases like fault detecting capability, mutant killing score, and execution time of test cases, testing cost, test case design efforts, requirement coveragebility and so forth. Above discussion concludes that test case optimization problem is not a single objective but a multi-objective problem. Moreover, though soft computing techniques are being explored in this area, but there is still a good scope to implement these individual & hybridization of intelligent techniques in future to optimize the test cases by considering it as a multi-objective optimization problem.

4 Conclusion & Future Work

Test data generation is one of the key issues in software testing. A properly generated test suite may not only locate the errors in a software system, but also help in reducing the high cost, efforts associated with software testing. Present work surveyed various techniques of software test case optimization. First we summarized traditional and advanced test optimization techniques, and then we identified gaps in existing techniques. Optimization of test cases is multi-objective optimization, NP-complete and peculiar nature problem. Soft computing can be used for these type problems, whose inexact solutions driving is computationally-hard tasks such as the solution of "NP-complete problems".

In conclusion, a lot of test cases optimization techniques have been developed for achieving software testing effectiveness and fault coverage. Review of existing literatures has identified that there are several objectives of test case optimization like maximum number of defect detecting capability, minimum test design efforts/cost, minimum execution cost, maximum coveragebility of client requirements & codes, maximum mutant killing score and so forth. Therefore optimization of test cases should be treated as multi-objective optimization problem. However most of test cases optimization approaches are single objective. Single objective formulation of test cases optimization problem is not justified and not meeting the objectives of testing. Some objectives are conflicting in nature, coveragebility of one objective will suffer other objective while considering all objectives concurrently. So, there is strong need to shift the paradigm from single objective test case optimization to multi-objective test case optimization. Moreover for these techniques, soft computing approaches like Genetic Algorithms, Fuzzy Logic, Artificial Neural Network etc may be well suited for experimentation and validation purpose.

References:

[1] [Alberts, 1976] Alberts, D. S., "The economics of Software", Proceedings of National Computer Conference on quality assurance, held at Montvale, N.J., Vol. 45, pp: 433-442 , 1976.

[2] [Korel, 1990] Korel, B., "Automated test data generation", IEEE Transactions on Software Engineering, Vol. 16(8), pp: 870- 879, 1990 .

[3] [Dalal, 1996] Dalal, S., Cohen, D., Parelius, J. and Patton, G., "The Combinatorial Approach to Automatic Test Generation", IEEE Software, Vol. 13(5), pp: 83-87, 1996.

[4] [Yilmaz, 2004] Yilmaz, C., Cohen, M. B., Porter, A., "Covering arrays for efficient fault characterization in complex configurations spaces", the Proceedings of Int. Symposium on Software Testing and Analysis, ACM Sigsoft, Vol. 29(40), pp: 45-54, 2004.

[5] [Frankl, 1993] Frankl, P. G., and Weyuker, E. J., "A Formal Analysis of the Fault Detecting Ability of Testing Methods", IEEE Transactions On Software Engineering, Vol. 19(3), IEEE press, 1993.

[6] [Michael, 1997] Michael, C. C., Voas, J. M., "Problem of accuracy in prediction of software quality form directed test", Proceedings of the Italian Conference on Theoretical Computer Science (ICTCS), pp:1-12, 1997.

[7] [Hong Zhu, 1995]Hong Zhu "Axiomatic assessment of control flow-based software test adequacy criteria" Software Engineering Journal, September 1995,pp:194-204, 1995.

[8] [Haruka, 2008] Haruka, N., Fraunhofer, R. E., "Strategic usage of test case generation by combining two test case generation approaches", Proceedings of The Second IEEE International Conference on Secure System Integration and Reliability Improvement, pp: 230-235, 2008.

[9] [Xiao Qu, 2007] Xiao Qu, Myra B. Cohen, Katherine M. Woolf, "Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization", published in ICSM, IEEE ICSM 2007, pp:255-264, 2007.

[10] [Wang, 2007] Wang, X., Qin, Z., Han, F., "UML Based Hybrid Model for Generation of Software Reliability Test Cases", Journal of Xi'an Jiaotong University, Vol. 41(4), pp: 421-425, 2007.

[11] [Wong, 1997] Wong, W. E., Horgan J. R., Mathur, A. P., and Pasquini, A., "A test set size minimization and fault detection effectiveness : A Case study in space application", Proceedings of IEEE 21st Annual International Computer Software and Application Conference (COMPSAC-97) held at Los Alamitos, CA, IEEE Press, pp: 552-528, 1997.

[12] [Tallam, 2005] Tallam, S. and Gupta N, "A Concept analysis inspired greedy algorithm for test suite minimization" the sixth ACM SIGPLAN-SIGSOFT Workshop on program analysis for software tools and engineering, held at New York, NY, ACM Press, pp: 35-42, 2005.

[13] [Zeller, 2002]Zeller, A. and R. Hildebrandt, "Simplifying and isolating failure-inducing input," IEEE Transactions on Software Engineering, vol. 28(2), pp. 183–200, February 2002.

[14] [Phil, 2004]Phil McMin, "Search-based Software Test Data Generation: A Survey", Proceedings in Software Testing, Verification and Reliability, Vol. 14, pp:105-156, 2004.

[15] [Mala, 2010] Mala, D.J., Mohan, V., "Quality Improvement and Optimization of Test cases– A Hybrid Genetic Algorithm Based Approach", ACM SIGSOFT Software Engineering notes, Vol. 35(3), pp: 1-14, ACM Press, 2010.

[16] [Mohan, 2007] Mohan, V., Mala, D. Jeya, "IntelligenTester – Software Test Sequence Optimization Using Graph Based Intelligent Search Agent", Proceedings of IEEE International conference on Computational Intelligence and Multimedia Applications, Vol. 8(7), pp: 22-27, 2007.

[17] [Bendt, 2005] Berndt, D. J., and Watkins, A., "High Volume Software Testing using Genetic Algorithms", Proceedings of 38th IEEE International Conference on System Sciences, held Hawaii, Vol. 8(5), pp: 1-5, 2005.

[18] [Maxwell, 2000] Maxwell, P., Hartanto, I. and Bentz, I., "Comparing Functional and Structural Tests", Proceedings of International Test Conference, pp: 400-407, 2000.

[19] [Maity, 2005] Maity, S., Nayak, A., "An Improved Test Generation Algorithms for Pair-Wise Testing", Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), IEEE Press, pp: 1-2, 2005.

[20] [Williams, 2000] Williams, A. W., "Determination of test configurations for pair-wise interaction coverage", Proceedings of 13th International Conference on the Testing of Communicating System (Test COM -2000), held at Ottawa, Canada, pp: 59-74, 2000.

[21] [Leon, 2003] Leon, D., Podgurski, A., "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases", Proceedings of the 14th International IEEE Symposium on Software Reliability Engineering (ISSRE'03), pp: 442-453, 2003.

[22] [Elbaum, 2002] Elbaum, S., Malishevsky, A.G., and Rothermel, G., "Test case prioritization: a family of empirical studies", IEEE Transactions on Software Engineering, Vol. 28(2), pp: 159-182, 2002.

[23] [Harrold, 1999] Harrold, M. J., Rothermel, G., Untch, R., Chu, C., "Test-case prioritization: an empirical study", Proceedings of the International Conference on Software Maintenance, pp: 179-188, 1999.

[24] [Black,2004] Black, J., E. Melachrinoudis and D. Kaeli, "Bi-Criteria Models for All-Uses Test Suite Reduction," 26th International Conference on Software Engineering, Edinburgh, Scotland, UK, 2004

[25] [Harrold, 1993] Harrold, M. J., Gupta R., Sofifa, M. L., "A Methodology for controlling the size of test suite", ACM Transaction on Software Engineering Methodology, Vol. 2(3), pp: 270-285, 1993.

[26] [Shin, 2007] Shin, Yoo and Mark Harman, "Pareto Efficient Multi-Objective Test Case Selection", Proceedings of the International symposium on Software testing and analysis (ISSTA), held at London, U.K, ACM press, pp: 140-150, 2007.

[27] [Wong, 1998] Wong, W. E., Horgan, J. R., London, S., and Mathur, A. P. , "Effect of test set size minimization and fault detection effectiveness", Journal Software Practice and Experience, pp: 347-369, 1998.

[28] [Sprenkle,,2004] Sprenkle, S., S. Sampath, E. Gibson, A. Souter, L. Pollock, "An Empirical Comparison of Test Suite Reduction Techniques for User-session-based Testing of Web Applications," Technical Report 2005-009, Computer and Information Sciences, University of Delaware, November 2004

[29] [Grindal, 2004] Grindal, M., Lindstrom, B, Offutt, A. J., and Andler, S. F., "An Evaluation of combination test strategy for test case selection", Technical report HS-IDA-TR-03001, Organized by Department of computer Science , University of Skovde. Sweden, 2004.

[30] [Kim, 2002] Kim, J. M., Porter, A.,"A history-based test prioritization technique for regression testing in resource constrained environments", Proceedings of the 24th International Conference on Software Engineering, held at Orlando, FL, pp: 119-129, 2002.

[31] [Kim, 2000] Kim, J. M., Porter, A., and Rothermel. G. "An empirical study of regression test application frequency", In Proc. of the 22nd lnt'l. Conf. on Software. Eng., pp: 126-135, Jun. 2000.

[32] [Varun,2010] Varun Kumar, Sujata, Mohit Kumar, "Test Case Prioritization Using Fault Severity", International Journal of Computer Sci ence and Technology (IJCST) Vol. 1(1), September 2010, pp:67-71, 2010.

[33] [Leon , 2000] Leon, D., Podgurski, A., and White, L.J. "Multivariate visualization in observation-based testing", Proceedings of the 22nd International Conference on Software Engineering (Limerick, Ireland, June 2000), ACM Press, pp:116-125,2000.

[34] [Fischer, 1981] Fischer, K., Raji, F., Chruscicki, A, "A Methodology for Retesting modified Software", Proceedings of IEEE National Telecommunication conference, Vol-6(3), held at Piscataway, NJ, IEEE Press, pp: 1-6, 1981.

[35] [Miller, 1988] Miller, E. F., "Advances in Automating Software Testing", Proceedings of Software Engineering Congress, pp: 202-212, 1988.

[36] [Benedusi, 1988] Benedusi, P., Cmitili, A., DeCarlini, U., "Post-Maintenance Testing based on path changes analysis", Proceedings of Conference on Software Maintenance , pp: 352-361, 1988.

[37] [Weiser, 1984] Weiser, M., "Program Slicing" in IEEE Transaction on Software Engineering, Vol. 10(4), pp: 352-357, 1984.

[38] [Agarwal, 1991] Agrawal, H., DeMillo R.A., Spafford, E H, "Dynamic Slicing in the presence of unconstrained pointers", Proceedings of ACM Symposium on Testing, Analysis & Verification(TAV-4), held at New York, NY, ACM Press, pp: 60-73, 1991.

[39] [Agarwal, 1990] Agrawal, H., Horgan, J R., "Dynamic Program Slicing", Proceedings of Conference on programming language Design & Implementation (PLDI-90), held at New York, NY, ACM Press, Vol. 25(6), pp: 246-256, 1990.

[40] [Ferrante, 1984] Ferrante, J., Ottenstein, K. J. and Warren, J. D., "Program Dependence Graph and it's use in Optimization", Proceedings of 6th Springer Colloquium on International Symposium on Programming, held at London, Springer Press, pp: 125-132, 1984.

[41] [Lei, 2005] Lei, Yong, and James, H. Andrews, "Minimization of Randomized Unit Test Cases", Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05) IEEE Computer Society, pp: 1-13, 2005.

[42] [Agrawal, 1994] Agrawal H., "Dominators, super blocks, and program coverage," 21st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, Portland, Oregon, 1994.

[43] [Agrawal, 1999] Agrawal H.,, "Efficient Coverage Testing Using Global Dominator Graphs," 1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Toulouse, France, 1999.

[44] [Marre, 2003] Marre M. and A. Bertolino, "Using Spanning Sets for Coverage Testing," IEEE Transactions on Software Engineering, Vol.29(11), pp:974-984, Nov. 2003.

[45] [Sampath, 2004] Sampath S., V. Mihaylov, A. Souter and L. Pollock "A Scalable Approach to User-Session based Testing of Web Applications through Concept Analysis", in proceedings of Automated Software Engineering, 19th International Conference on (ASE'04) Linz, Austria, September 2004,

[46] [Rothermel, 1998] Rothermel G., M.J Harrold, J. Ostrin, and C. Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites," International Conference on Software Maintenance, November 1998.

[47] [Horgan,1992] Horgan J.R. and S.A. London, "ATAC: A data flow coverage testing tool for C," in Proceedings of Symposium on Assessment of Quality Software Development Tools, pp: 2-10, May 1992.

[48] [Jones,2003] Jones J. A. and M. J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage," IEEE Transactions on Software Engineering , 29(3), pp:195-209, March 2003.

[49] [Graves,2001] Graves, T. L., Harrold, M. J., Kim, J. M., Porter, A.,and Rothermel, G. "An empirical study of regression test selection techniques", ACM Transactions on Software Engineering and Methodology 10, 2 (April, 2001), pp:184- 208, 2001.

[50] [Elbaum, 2000] Elbaum, S., Malishevsky, A.G., and Rothermel, G. "Prioritizing test cases for regression testing". Proceedings of the 2000 International Symposium on Software Testing and Analysis (Portland, OR, August 2000), pp:102-112, 2000.

[51] [Rothermel,, 1999] Rothermel, G., Untch, R., Chu, C., and Harrold, M.J. "Test-case prioritization: an empirical study". Proceedings of the 1999 International Conference on Software Maintenance (August, 1999), pp:179-188, 1999.

[52] [Rothermel,, 2001] Rothermel, G., Untch, R., Chu, C., and Harrold, M.J "Prioritizing test cases for regression testing", IEEE Transactions on Software Engineering 27, 10 (October 2001), pp:929-948, 2001.

[53] [Mohanty, 2010] Mohanty, R., Ravi, V., Patra, M. R., "The application of intelligent and soft-computing techniques to software engineering problems: A Review", International Journal of Information and Decision Sciences, Vol. 2(3), pp: 233-272, 2010.

[54] [Thomas, 2006] Thomas Weise "Global Optimization Algorithms–Theory and Application", second edition published by Thomas Weise, Licensed under GNU FDL, 2006.

[55] [Berndt, 2003] Berndt,D. J. Fisher, L. Johnson, J. Pinglikar, and A. Watkins, "Breeding Software Test Cases with Genetic Algorithms", Proceedings of the 36th (IEEE) Hawaii International Conference on System Sciences, Waikoloa, Hawaii, pp:1-10, Jan 2003.

[56] [Mohapatra, 2009] Mohapatra, D., Prachet, B., "Automated Test Case Generation and its optimization for Path Testing Using Genetic Algorithm and Sampling", Proceedings of International Conference on Information Engineering, pp: 643-646, 2009.

[57] [Prabahar, 2009] Prabahar, T., James, Godwin, Guru, Subramani S., "Intelligent Test Case Optimization Using Hybrid Genetic Algorithm" , the International J. of Math. Sci. & Engg. Appls. (IJMSEA), Vol. 3(1), pp: 191-208, 2009.

[58] [Baudry, 2002] Baudry, B., Fleurey, F., Jézéquel, Jean-Marc, Yves, L.T., "Automatic Test Cases Optimization using a Bacteriological Adaptation Model: Application to .NET Components'', IEEE 17th International Conference, pp: 253-256, 2002.

[59] [Panda, 2008] Panda, V.R., Satanik, A.B., "Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory", Proceedings of first IEEE International Conference on Emerging Trends in Engineering and Technology (ICETET), pp: 298-303, IEEE Press, 2008.

[60] [Dorigo, 1996] Dorigo, M., Maniezzo, V., Colorni, A., "The Ant System: Optimization by a Colony of Cooperating Agents", IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol. 26(1), pp: 29-41, 1996.

[61] [Dorigo, 1991] Dorigo, M., Maniezzo, V., Colorni, A., "Positive Feedback as a Search Strategy", Technical Report No. 91- 016, held at Politecnico di Milano, Italy, 1991.

[62] [Huaizhong, 2005] Huaizhong, L., Lam, C.P., "An Ant Colony Optimization Approach to Test Sequence Generation for State-based Software Testing", Proceedings of the Fifth IEEE International Conference on Quality Software (QSIC'05), IEEE Press, pp: 255-264, 2005.

[63] [Singh, 2010] Singh, Yogesh, Kaur, A., Suri, B., "Test Case Prioritization using Ant Colony Optimization", ACM SIGSOFT Software Engineering Notes , Vol. 35(4), pp: 1- 7, 2010.

[64] [Mala, 2009] Mala, D.J., Mohan, V., "ABC Tester - Artificial Bee Colony Based Software Test Suite Optimization Approach", International Journal of Software Engineering, Vol. 2(2), pp: 15- 43, 2009.

[65] [Shihab, 2000] Shihab, A. Hameed, "Framework For Intelligent Meaningful Test Data Generation Model-IMTDG", WOO, pp: 11-16, IEEE Press, 2000.

[66] [Crina, 2009] Crina, G., Ajith, A., "On A Class of Global Optimization Test Functions" in ICS AS CR, pp: 247-252, 2009

[67] [Ibrahim, 2006] Ibrahim, W., ElChouemi, A., Hesham, El-Sayed, "Novel Heuristic and Genetic Algorithms for the VLSI Test Coverage Problem", Vol. 3 (6), pp: 402-408, IEEE Press, 2006.

[68] [Wenyan, 2008] Wenyan Li., Wang Jiyi, Lin Gaomin, "Generating and reducing test case based on covering rough sets", Proceedings of the 5th IEEE International Conference on Software Engineering, held at Los Alamitos, CA, pp: 439-449, IEEE Press, 2008.

[69] [Junmin, 2008] Junmin, Y., Zemei, Z., Zhenfang, Z., Wei, D., Zhichang, Q., "Design of Some Artificial Immune Operators in Software Test Cases Generation", Proceedings of 9th International Conference for Young Computer Scientists(ICYCS) Vol. 8(2), Crown Copyright, pp: 2302-2307, 2008.

[70] [Swain, 2010] Swain, S. K., Mohapatra, D. P., Mall, R., "Test Case Generation Based on Use case and Sequence Diagram", International Journal of Software Engineering,Vol.3(2), pp: 21- 52, 2010.