CHAPTER 3

PROPOSED SOFTWARE TEST OPTIMIZATION FRAMEWORK

3.1 MOTIVATION

Software test optimization is more interesting due to several reasons. First, it reduces the total execution time of a test suite to test the SUT in limited time. Secondly, to improve the turn around time required between software revisions during regression testing. Finally, the generation of only effective test cases makes high likelihood of finding out errors in the SUT.

The literature study gives a lot of insight into the existing work on software test optimization problem. The observations made out of the study, have motivated this research work to propose a new framework for software test optimization. Some of the observations are given in this section.

The requirements driven test suite optimization proposed by (Beizer 1990) cannot be generalized since, requirements usually change during software development. Also, some of the requirements could have been missed and some of them may be misunderstood which leads to the assignment of wrong test cases to those requirements. When the object plus attribute combination is applied for test suite reduction, it won’t produce much higher level of reduction when compared to other algorithms. If the requirements space is large, then assigning the test cases to every individual requirement is hard and requires a lot of human intervention.
In the case of BINTEST algorithm proposed by (Beyeda and Gruhn 2003), there is no attention given for infeasible path identification. Since, infeasible paths will not have any test data associated with it, if we try to identify a test data by iteratively generating them, that will lead to exhaustive testing only. Similarly, if a sub-path of a path P1 is already covered by some other path P2, there is no method given to avoid the generation of similar types of test cases to cover those previously covered edges.

In model based test data generation for tools testing (Ingo et al 2007), existing tools cannot be tested with this approach, since it needs instrumentation in the software, which requires partnering with the tool vendor. When using third party generated code generators, the source models are represented in graphical representation which cannot be recognized by this model.

In program slicing based approach given by (Agrawal 1991), if slicing cannot reduce the size of the program efficiently, the methods proposed will depress, general statement, branch or path coverage. Since it requires much time and cost to slice any program, the most disadvantage of the proposed method is the low efficiency in terms of time. There is no specific approach proposed for calculating the coverage metrics of the slices. If some of the components are common to two or more slices, then overall coverage cannot be the summation of those of the subsystems.

The predicate based slicing approach proposed by (Forgacs et al 1997), the most important observation is on deciding some predicates as influencing predicates. Since, the influencing factor of the predicates is not static which may be changing dynamically, some predicates which seem to be influencing at one point of time may not influence at some later point of time. Hence the decision of slicing based on influencing predicates is really in need of a lot of training and intelligence to solve it.
In the case of Bounded Exhaustive Testing (BET) proposed by David et al (2005), have several drawbacks which have been identified by them. If the specifications themselves have faults, then this approach will not work. The approach is an incomplete work. The use of BET cannot be assessed in software systems to ensure statistical reliability. Executing the test cases is again a problem. The TestEra tool that they used has problems in generating integer based test inputs.

In the AI planning based test case generation technique proposed by Peter and Johannes (2000), the procedure can however not prevent that some manual additions have to be made such as the expected system responses have to be added to the test sequence manually to yield complete test cases; although they derived some constraints on the test inputs automatically, the concrete test data still has to be defined manually. Besides, systematic consideration should be given to other aspects of the specification, e.g. performance and frequency requirements.

In neural networks based software test optimization (Meenakshi 2002), the drawbacks are its black-box data processing structure and, in some cases, a slow convergence speed. Thus, the data processing mechanism of a NN cannot be easily programmed, trained, understood, or verified in terms of rules. As pruning and sorting requires a lot of time for making the neural network to work properly, the other tasks such as clustering, converting from continuous to discrete and rule matching will make the approach to be highly time consuming. Also, the number of layers in the neural network is not properly specified. The rule engine has to be generated manually and is purely depending on the application under test. Hence there is no generalization achieved in this approach. If the rules are not defined accurately, the test case selection procedure will fail. Again this approach is requirements based.
In Fuzzy Logic based test data generation (Zhang 2007), there is no specific mechanism given for software test optimization. There is no method provided for calculating the probability which is used to handle uncertainty.

In Tabu Search based approach proposed by (Díaz et al 2008), more amount of memory is required in terms of long term memory to avoid stuck up at local optima and short term memory to remember all the test cases in the current search. Even though this short term memory is deleted often, this will lead to a crucial problem of effective utilization of memory.

In applying Genetic Algorithms (Pargas et al 1999), we have the risk of suboptimal solution, delayed convergence and strike up at local optima. As per Benoit Baudry et al (2005), the experiments with genetic algorithms were not satisfactory. The mutation rate has to be increased consistently when compared to usual application of genetic algorithms (mutation rates lower than 2%). With 2% of mutation rate, the best mutation score was 80%, after a long computation. Moreover, the results are not stable, the convergence is slow and one population can be more efficient from the following, due to a non-explicit memorization.

Even though, the resultant test suite using Ant colony optimization (ACO) proposed by Li and Lam (2004) contains minimum number of test sequences, the test case within the test suite has longer sequence length which violates the optimality requirement criterion. Similar observation can be drawn when the number of ants exceeds three. The algorithm proposed by them, initially had an assumption that the two ants are started at an initial node, and during random selection of next node, they will go to the same node. The problem here is that, due to the random selection, we cannot expect that the approach will always produce the same node to be selected by two different ants at different instances of time. In our earlier works on test suite optimization, we employed multi-agent based approach in the test optimization process.
Bacteriologic algorithm based test case optimization approach proposed by Baudry et al (2005), has been applied on some test benches. Since BA doesn’t have cross over operator, the further generation of test cases loses the advantage of combining the best properties of both the parents.

In the work of (Dhavachelvan and Uma 2003), a test agent system design was proposed. In which they employed intelligent agent techniques to provide active assistance to the tester. The work concentrated on generation of test cases for the given SUT. It includes rules for construction of test cases and elimination of test cases. Even though the approach has relevant advantages, it doesn’t provide any way for optimization of test cases.

Hence, these observations have motivated this research work to develop a new software test optimization framework which comprises of the advantages of all of the discussed approaches and at the same time minimizes the drawbacks of them consistently. The proposed framework is a hybrid optimization framework composed of three different ways of achieving software test optimization using Intelligent Agents.

3.2 PROPOSED HYBRID TEST OPTIMIZATION FRAMEWORK

The objective of the thesis is to develop a novel test optimization framework by applying hybrid intelligence based search approaches. The scope of the research work focuses on the problems in existing test optimization approaches such as Ant Colony Optimization (ACO), Genetic Algorithm (GA) and Bacteriologic Algorithm (BA) and tries to achieve test optimization by means of three ways. Test Sequence Optimization, Test Case Optimization and Test Suite Optimization. The proposed framework is evaluated based on the test adequacy criteria such as State Coverage, Branch Coverage, Path Coverage and Mutation Score. The overall framework of the proposed approach is shown in Figure 3.1.
In the proposed framework, the first approach applied intelligent graph based searching by applying both path finding and constraint satisfaction aspects of agents to find the efficient few feasible and non repeatable test sequences. The second approach applied the characteristics of genetic algorithm with local search (GA-LS) by means of a Hybrid Genetic Algorithm (HGA) to achieve quality test cases that are likely to reveal more number of faults. In addition, in the third approach, an Artificial Bee Colony (ABC) based optimization is applied for complete software test suite optimization.

Hence, the proposed novel test optimization framework includes three important optimizations:

1. Test Sequence Optimization
2. Test Case Optimization
3. Test Case and Test Sequence Optimization
3.2.1 Test Sequence Optimization

The intention of test sequence optimization is to reduce the number of test executions. It is assumed that, we have a set of test cases in the test case repository. The intention of software testing process is to execute these test cases against the software under test and record the test results. The tester wishes to find out the coverage of all the test paths, the statements and branches in the SUT. If the same set of paths is covered by more than one test case, then the time spent is wasted since it will not reveal any new errors during its execution. Test sequence optimization identifies an efficient set of execution sequences with identification of repetition of sub-paths and infeasible paths. The generated test sequences are optimal and the test cases are exercised against these test sequences which results in less testing time and cost.

If the test sequence optimization process is guided by human, then he/she can monitor the statements covered, the infeasible statements which cannot be covered by any of the test cases and the sub-paths which are simply a repetition of some other paths’ sub-paths. This is the reason why many industries still trust the manual testing process in spite of the existence of many automated testing tools in the market.

Even though human based test sequence optimization is efficient, if the SUT is large and complex this will take a lot of time and also consumes a large amount of testing resources. Research works indicated the loss incurred due to human errors. Hence, there is a need for an approach which couples human like intelligence in decision making process with automation in tools. This leads to the application of Intelligent Agents (IA) for performing the test sequence optimization process with human like intelligence performed in an automated way. The agent developed in this research work, is an Intelligent Search Agent (ISA) that searches the Software under Test (SUT) for the
generation of few efficient test sequences. These generated set of test sequences achieves both branch coverage and statement coverage based test adequacy criterion.

### 3.2.2 Test Case Optimization

The intention of test case optimization is to reduce the number of test cases that must be exercised against the SUT. Test case optimization focuses on generating and selecting a few efficient test cases from an infinite number of possible ones. Automated testing tools simply generate the test cases and enforce the execution of all these test cases and do not worry about selection of efficient ones to reduce the testing time.

The efficiency of a test case is determined by its ability to cover the software and to reveal as many errors as possible in the SUT. In test case optimization, a Hybrid Genetic Algorithm (HGA) based approach is applied to generate efficient test cases. During each generation, the quality of test cases is improved and finally only an effective few test cases were filtered that have higher coverage and high mutation score. The proposed Hybrid Genetic Algorithm (HGA) combines the features of Genetic Algorithm (GA) and local search (LS) with memorization. The proposed framework “Hybrid Tester”, applies Hybrid Genetic Algorithm (HGA) with two improvement heuristics to select the best parent test cases for generating efficient offspring in further generations.

### 3.2.3 Test Suite Optimization

The test suite optimization process involves generation of effective test cases in a test suite that can cover the given SUT within less time. The proposed framework called “ABC Tester” applies the artificial bee colony optimization (ABC) approach for test suite optimization. Artificial Bee
Colony (ABC) optimization is a non-pheromone based swarm intelligence approach motivated by the intelligent behavior of honey bees; the colony consists of three groups of bees: employed, onlookers and scouts.

The proposed approach based on Artificial Bee Colony (ABC) optimization represents each test case as a possible solution in the optimization problem and happiness value - a heuristic introduced to each test case corresponds to the quality or fitness of the associated solution. This is achieved by developing three agents namely Search Agent, Selector Agent and Replace Agent that mimics the behavior of employed, onlooker and scout bees in the bee colony. The agents work in parallel and hence achieve better software test suite optimization.

### 3.3 SUMMARY

In the light of the literature study, this thesis proposes a novel software test optimization framework namely “Hybrid Test Optimization Framework”. It has test sequence, test case and test suite optimization as the tasks. As the outcome of the research work, a prototype tool is developed using Java and is registered under Java Research License (JRL) available in the URL https://intelligentester.dev.java.net.

In the proposed framework, each of the test optimization approaches, have been developed with frameworks, algorithms and have been evaluated using case studies. Also, each of the proposed approaches have been compared with existing approaches and proved that, the proposed approaches produces optimal or near optimal solutions.