CHAPTER 1

OBJECTIVES AND OUTLINE OF THE RESEARCH

1.1 INTRODUCTION

Software testing is defined as the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component (IEEE Standard 610.12-1990). It is the execution of a test object with the intent of finding errors by a finite sample of test cases. Testing can show the presence of errors but not their absence (Myers 1979). Accordingly, the main challenge in testing is the design of error-sensitive test cases. This task ultimately determines the scope and quality of the testing process. And, testing is so far the only method that allows dynamic program behavior to be checked in the real application environment. Hence, software testing is used to find defects in the software; to ensure that the software meets all the requirements of the customer; to check whether the product meets functional and performance objectives; to ensure safety and regulatory compliance; and to ensure that the production standards are met.

Even though achieving zero-defect quality software is the ambition, it is not possible in reality (Pressman 2005). Software testing can be very costly and it typically consumes at least 50% of the total cost involved in software development (Beizer 1990, Ellims et al 2006, Michael Grottke 2007). As per the study of National Institute of Standards and Technology, the cost for an inadequate infrastructure for software testing is estimated to be from $22.2 to $59.5 billion (Tassey 2002). These studies indicated the need
for an optimization approach in the testing process to reduce the amount of resources (in terms of time, cost, man power, system components etc.) required for testing without compromising the quality. Hence, the importance of software test optimization has increased in recent years due to customers’ need for quick delivery of quality software, reduced software development lifecycle, changing markets with global competition and rapid development of new processes and technologies.

1.2 SOFTWARE TEST OPTIMIZATION PROBLEM

Because software and any digital systems are not continuous, testing boundary values are not sufficient to guarantee correctness. All the possible values need to be tested and verified, but complete testing is infeasible. Exhaustively testing a simple program to add only two integer inputs of 32-bits (yielding $2^{64}$ distinct test cases) would take hundreds of years, even if tests were performed at a rate of thousands per second.

As Phil McMinn briefed in his paper (McMinn 2004), exhaustive enumeration of a program’s input is infeasible for any reasonably-sized program, yet random methods are unreliable and unlikely to exercise deeper features of software that are not exercised by mere chance. This survey indicated the importance of applying optimization techniques in software testing without compromising the quality.

Since, manual testing is costly and error prone, automation is a good way to cut down time and cost to achieve optimization in the testing process (Gerlich 2007). But, software testing tools and techniques usually suffer from a lack of generic applicability and scalability. In general, significant amount of human intervention is still needed in automated testing. The degree of automation remains at the automated test script level. The problems associated with the common testing methods such as random
testing, specification based testing and path based testing have made optimization as a must in the testing process.

1.2.1 Random Testing

In random testing, the test case selection is simple and straightforward, in which the tests are randomly chosen from the space of possible inputs. The literature study indicates that random testing is more cost effective for many programs. But only very subtle errors can be discovered with low cost in random testing. Also, one cannot assure that, this testing approach can cover the SUT and achieves both coverage and fault revealing ability based test adequacy criteria.

1.2.2 Specification Based Testing

Consider that, there is a specification for calculating five types of commission and seven types of discount for a given product. Testing every possible combination of just commission and discount requires 35 test cases. Even if there are 20 factors, each taking only 4 different values, then a total of $4^{20}$ or $1.1 \times 10^{13}$ different test cases has to be examined. If a team of programmers could be found who could generate, run and examines test cases at an average rate of one for every 30 seconds, then it would take more than a million years to complete (Beizer 1990). Hence, specification based testing, can lead to exhaustive testing. Khurshid et al (2004) proposed an automated methodology called ‘TestEra’ to do specification based testing for Java programs. Due to the fact that, the specifications are changeable during program development, they alone cannot be used for generating test cases. They must be combined with other techniques to achieve better results.
1.2.3 Path Based Testing

Common form of testing to code requires that each path through the module must be executed at least once. The most powerful form of structural testing is path coverage. But in a module that has a number of loops, testing every possible combination of checking condition will take a lot of time to complete. It is also not possible to do this (Agrawal 1994).

\[
\text{Loop } \leq 18 \text{ times}
\]

![Diagram showing path based testing with loop]

**Figure 1.1 Path based testing with loop**

In the Figure 1.1, there are $10^{12}$ possible paths. There are five possible paths through the loop, and the total number of possible paths is $5^1 + 5^2 + \ldots + 5^{18} = 4.77 \times 10^{12}$. This indicates that, the huge number of paths renders exhaustive testing to code as infeasible as exhaustive testing to specifications. Hence the selection of few effective test cases that exercises all possible executable blocks of the given software is the goal to achieve.

The complexities involved in the common testing methods, as described, forces the need for optimization in the testing process without compromising the quality. Realistically, testing is a trade-off between budget, time and quality. It is driven by profit models. The pessimistic and unfortunately most often used approach is to stop testing whenever some or any of the allocated resources such as time, budget, or test cases is exhausted.
The optimistic stopping rule is to stop testing when either quality meets the requirement, or the benefit from continuing testing cannot justify the testing cost. This will usually require the use of test adequacy criteria to evaluate and predict the quality of the software under test.

1.3 PROBLEM COMPLEXITY

Software testing is a dynamic activity in which test input generation and execution can be seen as optimization problems. The input domain of most of the programs is likely to be very large. A cost-surface could be formed by applying the cost function to every possible program input. It is this surface which is effectively being searched when attempting to generate test data.

Given the complexities of software systems, it is extremely unlikely that this cost surface would be linear or continuous. The size and complexity of the search space therefore limits the effectiveness of simple gradient-descent or neighborhood searches as they likely to get stuck in locally optimal solutions and hence fail to find the desired test data.

Ultimately, the cost function used for optimization is typically related to the cost and time needed for the testing process, the size of the test suite and the coverage or fault revealing capability of the test cases.

Since, software testing is a combinatorial problem in which finding solutions at a reasonable computational cost (Harman et al 2007) is highly complex, it has been classified as NP-complete or NP-hard, or be a problem for which a polynomial time algorithm is known to exist but is not practical.

Hence, software test optimization problem is devised as a multi-objective optimization problem and the overall objective function is to
minimize the total cost and time needed for the testing process and to maximize the various test adequacy criteria (coverage based, mutation score/fault revealing ability based etc.) to achieve quality.

It is formally given as,

Max.

\[
\begin{align*}
\text{Coverage (SUT)} & \quad (1.1) \\
\text{Mutation Score (SUT)} & \quad (1.2)
\end{align*}
\]

Min.

\[
\begin{align*}
\text{Cost (Testing Process)} & \quad (1.3) \\
\text{Time (Testing Process)} & \quad (1.4)
\end{align*}
\]

1.4 SOLUTION METHODOLOGY

A large number of approaches for test data generation and optimization in software testing have been reported with varying degrees of success. In recent years, the application of artificial intelligence (AI) techniques in software testing to achieve quality software is an emerging area of research that brings about the cross fertilization of ideas across two domains such as AI and Software Engineering (Briand 2002, Pedrycz et al 1998).

The literature review shows that the research on software testing problems has centered mostly on software test optimization (Eric et al 1999, Xiao et al 2007, Zuohua et al 2008, Chen et al 2003) and intelligent software testing techniques are extensively proposed to solve software test optimization problems. Intelligent software testing techniques support quality assurance by gathering information about the software being studied (McMinn 2004).
These include optimization approaches such as simple heuristics, artificial intelligence (AI) techniques, population based search heuristics, swarm intelligence based approaches and neighborhood based search heuristics. But, there is still a need for a more effective solution approach to more general problems such as test sequence, test case and test suite optimizations and it is essential to develop models and efficient algorithms to achieve these optimizations by satisfying the specified test adequacy criteria.

Figure 1.2 shows the classification of some of the approaches used in software test optimization. Knowledge based approaches are designed to find good approximations to the optimal solution in large complex search spaces. They are subdivided into heuristics based approaches and AI based approaches. The shaded boxes in the Figure 1.2 show the approaches applied in this research work, which forms the proposed test optimization framework.

AI based approaches include Expert Systems, Intelligent Agents (IA), Neural Networks (NN) and Fuzzy systems. Fuzzy logic and Neural Networks (NN) and other approximation methods such as Chaotic Neural Networks have been proposed for software test optimization over the past years (Meenakshi et al 2002, Zhang et al 2007, Wang 2004, Cao 2006 and Wang 2009). Among them, the intelligent agents can be implemented with sophisticated intellectual capabilities such as the ability to reason, learn, or plan (Russell 2005). This means that the underlying agent architecture must support sophisticated reasoning, learning, planning, and knowledge representation (Russell 2005). Intelligent Agents have been applied to various applications to achieve efficient results, and have been emerged as potential techniques to provide solutions with human like intelligence and exhibits properties such as autonomy, social ability and interoperability (Dhavachelvan et al 2003, Mangina 2005, Miao et al 2007 and Zhiyong et al 2008).
• Meta-heuristic search techniques are high-level frameworks which utilize heuristics in order to find solutions to combinatorial problems at a reasonable computational cost (Micheal 1999, Harman 2007). They have been classified into neighborhood based search approaches and population based search approaches. Several meta-heuristic search techniques such as applying rules, meta-heuristics (like Genetic Algorithm (GA), Ant Colony Optimization (ACO), Tabu Search, Simulated Annealing, Bacteriologic Algorithm (BA), etc.), have been proposed for some specific types of problems in test optimization (Hélène et al 2007, Pargas et al 1999, Li and Lam 2004, Díaz et al 2008, Baudry et al 2005).

• Neighborhood based search approaches include Tabu Search (TS), Simulated Annealing (SA), Hill climbing Algorithm and Chaotic Simulated Annealing techniques (Aihara 2002, Wang 1998). Simulated Annealing is an optimization technique that has been applied for solving a wide range of combinatorial optimization problems, in which the feasible solutions represents different states of the problem. In Tabu Search based approach, two separate lists are maintained to attain near global optimal solution (Tracey 1998). Hill Climbing is used to improve a solution, with an initial solution randomly chosen from the search space as a starting point. The neighborhood of this solution is investigated. If a better solution is found, then this replaces the current solution. This process is repeated till either the desired solution is found or the specified number of iterations is reached (McMinn 2004).
Population based search heuristics, which belongs to the random search category, guarantees near optimal solutions in actual cases. The popularly known population based search heuristics are Genetic Algorithm (GA), Bacteriologic Algorithm (BA), Hybrid Genetic Algorithm (HGA) and Particle Swarm Optimization (PSO) such as Ant Colony Optimization (ACO) and Artificial Bee Colony Optimization (ABC). These approaches are useful for any hard optimization problem. Over the past thirty years, there has been a growing interest in problem solving systems based on the principles of evolution and heredity. Such systems maintain a population of potential solutions with selection processes based on the fitness of individuals, and some recombination operators. Among them, Hybrid Genetic Algorithm that belongs to population based approach combines the features of Genetic Algorithm with Local Search (GA-LS). It has been applied for many hard optimization problems (José et al 2002, Yici et al 2005, Wappler et al 2006, Kim et al 2007, William et al 2008, Zhao et al 2009) and produces optimal or near optimal solutions. Also, Artificial Bee Colony optimization that belongs to non-pheromone based swarm intelligence algorithms is considered suitable for many optimization based problems (Karaboga et al 2008, Mohammad et al 2007, Alok Singh 2009) in producing optimal results to NP-hard problems.

The above discussion indicates that, artificial intelligence based approaches and population based search heuristics are useful tools for software test optimization problem.
1.5 OBJECTIVES OF THE THESIS

The major observations derived as part of the literature study, show the problems associated with existing approaches such as NN, GA, BA, ACO, TS and SA. Some of them are: the black box data processing structure and slow convergence speed of Neural Networks (NN), the acceptance of inferior solutions early in the search process with relative freedom of Simulated Annealing, excessive amount of memory requirement of Tabu search, non-explicit memorization of best individuals, slow convergence and non-stable results (strike up at local optima) of Genetic Algorithm (GA), memory requirement and absence of crossover operator in Bacteriologic Algorithm (BA), higher length test sequences, repetition of nodes within the same sequence and problems involved in initial assumption of Ant Colony Optimization (ACO), violate the optimality requirement criterion.

Concerning the above, the objective of the thesis is to propose a test optimization framework that addresses the problems in the testing process, problems in the application of existing knowledge based approaches in test optimization and problems involved in manual and automated testing. The focus is now on identification of other knowledge based approaches to solve test optimization problem effectively.

Further literature study, exposes the application of other knowledge based approaches such as Intelligent Agents, Hybrid Genetic Algorithms and Artificial Bee Colony Optimization to various NP-hard problems to attain optimal or near optimal solutions.

In the light of the above consideration, this thesis proposes a novel software test optimization framework shown in Figure 1.3, using hybrid intelligence based search approaches such as Intelligent Agents, Hybrid Genetic Algorithm and Artificial Bee Colony Optimization to provide an
optimal and near-optimal solution to achieve test sequence, test case and test suite optimizations respectively. The proposed approaches have been evaluated based on the test adequacy criteria such as State Coverage, Branch Coverage, Path Coverage and Mutation Score (fault revealing ability).

**Figure 1.3 Methodologies applied in the Proposed Software Test Optimization Framework**

As the outcome of the research work, a prototype tool called “IntelligenTester” is developed and is registered under Java Research License (JRL) and its description is available in the URL https://intelligentester.dev.java.net.

### 1.6 CONTRIBUTIONS OF THIS THESIS

The major contribution of this thesis is that, the proposed framework is implemented as a tool namely “IntelligenTester” using Java. The tool provides a complete automated framework to achieve software test optimization. Other major contributions of this work include optimization problem formulation, development of decision making algorithms for each of the approaches and experimental assessment using empirical data.
1.6.1 Optimization Problem Formulation

For each of the proposed approaches, objective functions are formulated. In test sequence and test suite optimizations, the objective function is to maximize the fitness function. In test case optimization, a multi-objective function is devised in which the first objective function is to maximize the test adequacy criteria of each of the test case and the second objective function is to minimize the size of the test case set.

1.6.2 Development of Decision Making Algorithms

To achieve test optimization, heuristics guided search algorithms are proposed for each of the approaches. For test sequence optimization, two algorithms namely infeasible test sequence identification algorithm and repeated subsequence detection algorithm are proposed with heuristics Happiness Value and Similarity Measure respectively. For test case optimization, a Hybrid Genetic Algorithm (HGA) with improvement heuristics RemoveSharp and LocalOpt is developed. For test suite optimization, Artificial Bee Colony optimization algorithm with improved fitness function based on Happiness Value heuristic is applied.

1.6.3 Empirical Study

In this research work, an empirical study is conducted to explore the application of artificial intelligence and evolutionary computation techniques in software test optimization. Especially, the usefulness of Search Based Software Engineering (SBSE) using intelligent approaches was thoroughly analyzed and new techniques were proposed for software test optimization. Based on the literature survey and to the best of the gathered knowledge, the proposed approach is a novel approach, since no other research works have so far applied Intelligent Agents (IA), Hybrid Genetic
Algorithm (HGA) with improvement heuristics and Artificial Bee Colony (ABC) approach for software test optimization. The approaches have been evaluated using the test adequacy criteria such as coverage and fault revealing capability of test cases.

1.7 OUTLINE OF THE THESIS

In this chapter, the scope and objectives of the research work, the application of various knowledge based approaches to software test optimization and solution techniques used in this thesis are discussed. The rest of the thesis is organized as follows:

- Chapter 2 provides the literature survey on software test data generation and optimization. It discusses how existing traditional, algorithmic and knowledge based approaches were applied in software test optimization problem. Also it provides the study on the application of the proposed search approaches in solving other NP-hard problems.

- Chapter 3 demonstrates the proposed test optimization approach. It starts with the overall framework for the test optimization approach and proceeds to explain the proposed approaches for software test optimization.

- Chapter 4 provides a discussion on the proposed software test sequence optimization framework using Intelligent Search Agent (ISA). It shows the internal architecture of the proposed agent with algorithms for infeasible test sequence identification and repeated subsequence detection. The performance comparison of the proposed algorithm for various case studies is also presented.
Chapter 5 focuses on the proposed software test case optimization framework using Hybrid Genetic Algorithm (HGA). It begins with test case optimization using evolutionary algorithms such as the Simple Genetic Algorithm (SGA) and Bacteriologic Algorithm (BA). Then it proceeds with the discussion on the proposed “Hybrid Tester” framework based on a Hybrid Genetic Algorithm (HGA) with improvement heuristics ‘RemoveSharp’ and ‘LocalOpt’ to achieve near global optimal solution in test case optimization. The performance comparison of the proposed algorithm for various case studies is also presented.

Chapter 6 explains the proposed software test suite optimization framework using Artificial Bee Colony Optimization (ABC). The chapter provides an insight into Artificial Bee Colony (ABC) algorithm and then applies the algorithm for test suite optimization. The approach uses three agents namely Search Agent, Selector Agent and Replace Agent which work in parallel to do test suite optimization within less time and cost. The performance comparison of the proposed algorithm for various case studies is also presented.

Chapter 7 gives the results and discussions based on the performance analysis of the three approaches.

A summary of the present research work and an indication of future research directions are presented in the concluding Chapter 8.