CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The subject of this thesis is test suite optimization for effective test execution. Software Testing is the process of executing a software application in the motive of identifying errors or to confirm whether the actual results matches the expected requirements and to ensure the software developed is defect free. It also helps to identify gaps or missing requirements which differs from the actual requirements. It can be either done manually or by automated testing tools. The literature has many definitions for software testing. As per the ISO/IEC and IEEE standard, a dictionary of software engineering ISO/IEC/IEEE 24765-2010 defined, testing is “An activity in which a component or system is performed under specific conditions and environments, and the results are observed or noted, and a valuation is made based on some features of the components or system. Test process of a software process includes preparation activities, actual testing work and reporting the test results. Figure 1.1 shows the general software development life cycle process.

![Figure 1.1 General SDLC Process](image-url)
1.2 SOFTWARE COMPONENT TESTING

Software module or software component testing is comparatively active space of research in software engineering, which reflects the increased number of usage of components. The British computer society defined certain standards for Software Testing. The software component testing standard defines, software component as a smallest software element, for which a separate requirement is provided. The practice of executing a software with the motive of fault detection is one of the popular definition of Software testing. Software testing concentrates on the performance evaluation of software quality aspects, such as performance, accuracy, security and consistency. In general the process of software testing is, selection of appropriate rules as test cases to be run against the SUT, which is assessed based on the knowledge and skills of testers. The result of the software under test is compared with the expected results of software requirement specification document provided by end client or customer. If the results does not matches, then it is understood that a failure or error has occurred and the software is suspected to have a defect. Usually, the test result from software execution relies on both the current software state and the input value. The state of the software is, the aggregate values of all the variables put in registered memory at the same time. As a software module may have many probable inputs and states, it is not possible for complete testing with all combinations of given input values and states. Consequently, defects can often remain unidentified even after testing, which may cause to product failure or error after releasing the software.

1.3 TESTING RESEARCH IN GENERAL

The motive of software development processes are, to deliver a software that meets the required functionalities with standard quality within the given budget and delivered on time. These features are essential to the end product, since if any of these attributes like functionality, money, quality, and timing is
not handled in appropriate manner, then economically the software is possible to fail. The software development is generally a trade-off among these mentioned four attributes (Kaner C et al, 1999). From this view, there is no surprise that the research in software testing is used to enhance the processes for better coverage in requirements perspective, or to reduce the effort and cost of testing work by also retaining its existing quality.

The anticipated aims for the research in software testing are to achieve more efficiency in testing, test-based modelling, 100% automation and extensive test concepts (Bertolino A, 2007). The improvement in testing efficiency would relates that the test practice could be executed on its maximum efficiency with optimized testing methods and smart tools, to achieve better quality (Harrold M.J, 2000). The second listed motive is complete test automation, which focuses to build a highly developed test automation system to do entire self-governing testing work. However, this motive is difficult to be accomplished, because though the automation level is high, the application still may need manual intervention to ensure results or at-least to set up and sustain the system (Bach J, 1997). The next desired objective is test-based structuring that focuses at evolving the software towards modelling processes which provide easier and widespread support for testability. The difference between model-based testing and test-based modelling is in the location; test-based modelling develops the models based on testability, whereas model-based testing tests the software or application using the defined model. The fourth objective of the universal test theory concentrates on building an inclusive, consistent, and thorough framework to evaluate and compare the potential and limitations of different testing methods. These focused objectives might not be practical in short term, but all of them focuses at one main motive, that is making the testing easy. The effect of researches in software engineering and software enhancement management is explained in several articles by many authors. In software engineering, the effect
of scholarly reviews have been studied which are related to the current software market. Based on the outcomes, it looks like that the research in software engineering and any other software field have a close link. The fundamental system structure and new insight from the academic world explains how the market getting affected by the new technologies and developments. Figure 1.2 shows the general software development life cycle process.

![Software development life cycle](image)

![Software testing process](image)

**Figure 1.2 General Software Testing Process**

However, as evaluated, it may take many years for the industry to accept and implement the observed concepts. On the other hand, the current state of software engineering and software testing researches may still be a whole new concept for a real world software organization. In fact, even though the surveys
and studies of software engineering have got enhanced for over 3 decades, there are yet many fields that should be observed more in detail. The conclusion is that the understanding of software testing process is yet inadequate, and that over fifty percent of existing methods are based on assumptions and not coincides with formal facts that would provide only replicate results. It is concluded that an approach to create the groundwork for developing test scheme is to generate an experiential understanding to be aware of which attributes or features can elucidate when the complications occurs.

In the research of software testing, the experimental studies should be done to learn real-world happenings. The software engineering schemes are one among the complex systems that have been created so far. As per the current improvements in experiential software testing reviews, metrics are collected by applying the new approach to software engineering journals. The report is between the later of 1990’s and earlier of 2000s. On software engineering, roughly around 1,800 empirical studies have been conducted on software engineering and testing field. Among them, approximately 20% of the published journals were in a scientific field. The most general concepts of software testing studies in this decade were related to different techniques and methods, tools, inter computer communication, production quality and software life cycle (Sjoberg D.I.K et al, 2007).

The knowledge implementation in software testing field is not as simple as in various other fields. For instance, adopting one testing method for various projects may give various results. In software engineering and testing, the basis of obtaining information is repetitive; the theory is based on the wisdom of available methods, tools and techniques, but the original concept keep changing based on the collection of data and assessments in the real test world. The imperative part is to persistently question and advance the hypothesis until it elucidates the observed occurrence (Juristo N et al, 2001). Though few of the
metrics or result measures are nebulous or inaccurate or some hypothesis may only clarify the partial occurrence, it is good to have a biased perceptive that can help for future concepts than to get rid of an outcome just because it does not give explanation or cover all expectations (Pfleeger S.L, 1999).

1.4 FUNDAMENTAL APPROACHES TO SOFTWARE TESTING

Software testing techniques have been categorized on the basis of their contrasting fundamental approaches: static vs dynamic testing, and black-box vs white-box testing. In static testing, the software specification and or program code is examined to identify fault without executing the program; while in dynamic testing, faults are revealed by running the program on a computer and observing its behaviour. In black-box testing, test case are based on functional and other specified requirements of the software without considering the details of the program code; while in white-box testing, test cases are based on knowledge of the program code and design (Boris Beizer, 1990). These fundamental approaches to software testing categories are represented as a quadrant diagram in Figure 1.3.

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black-box</td>
<td>Black-box</td>
</tr>
<tr>
<td>White-box</td>
<td>White-Box</td>
</tr>
</tbody>
</table>

Figure 1.3 Fundamental software testing approaches

Each of the quadrants in Figure 1.3 may be considered as important area of software testing research in its own way, for example the upper-left-hand
quadrant in figure can be characterized as static black-box testing in which the functional and other specified requirements of the software are examined without executing the program. However, the focus of the present research falls within the upper right quadrant and is categorised as dynamic black-box testing. Within the category of dynamic black-box testing, two further contrasting fundamental approaches can be identified; positive vs negative test. Positive testing employs only valid input values in order to demonstrate that the software conforms to its specifications, whereas negative testing employs invalid inputs to test software robustness. Software testing will generally include both positive and negative testing.

1.5 IMPORTANCE OF SOFTWARE TESTING

Testing is important because software failures or defects could be expensive or even dangerous. Software bugs can potentially make financial and human loss as software history has such examples. In April 2015, Bloomberg terminal in London crashed because of software glitch which affected more than 300,000 traders on monetary markets. It required the government to postpone a 3bn pound debt sale. Nissan cars had to summon up over 1 million cars from the market due to software malfunction in the airbag sensor detectors. There have been reported two accidents due to this software failure. Starbucks was required to close around 60 percent of their stores in the U.S and Canada due to software issues in its POS system. At one point they had to serve the coffee for free as they were unable to process the transactions. In 2015, fighter plane F-35 fell down due to a software bug, making it incapable of detecting targets correctly. China Airlines Airbus A300 crashed because of a software bug on April 26, 1994, killing 264 innocent lives.
1.6 ADVANTAGES OF SOFTWARE TESTING

Software testing helps to identify the defects before deployment. It also reduces incompatibility and interoperability issues. Testing ensures quality and effective performance of the software or application. Sometime the newly developed system has to integrate and function with existing systems. In such cases testing gives assurance that it performs well on its own and also does not impact the existing system due to the integration. During testing itself the possible areas of software improvements are detected. Creation of defect reports during testing helps the project managers for preparing progress reports, forecasting the hardware requirement in future for similar projects. Software testing is the proof that it meets the necessary business requirements. It also helps to reduce penalties for errors, risk of failure and that in turn reduces the maintenance resource cost. By the errors and failures Customer’s reliability and satisfaction get improves. The quality always helps to stay in the competitive field and wins businesses and good-will.

1.7 TESTING ARTIFACTS

Testing techniques can be based on pre-code artifacts such as design, requirements and architecture specifications. Techniques that uses these pre-code specifications for tasks such as test-planning and development, can help to improve the overall testing process. The software testing process can produce several artifacts.

1.7.1 TEST PLAN

A test specification is called as a test plan. In general it is a document that describes the scope and activities of testing. It is the basis for any software testing or product testing. It also describes the approach, resources, time and schedule of intended test activities. A test plan that addresses only one test phase
is known as Phase test plan whereas a test plan which addresses multiple test levels are called as Master test plan. The format of the test plan varies based on the process, standards followed, however there are some general expectation on what are all the information a test plan should contain. Those are a test plan identifier, which is a unique identifier for the document, introduction of the plan like the overview, objectives or goals and constrains, references to the related documents, project and configuration management plan, list of test items, list of features to be tested, which approach should be followed for testing, testing level, testing activities that must be redone when testing is resumed, test deliverables such as test scripts, test reports, test environment, cost and effort estimate of testing, schedule, staffing and training needs, roles and responsibilities of each member, Risk and mitigation plan, assumption and dependencies and any approvals from authorities.

1.7.2 TRACEABILITY MATRIX

A Traceability matrix is a table that maps the design documents or requirements to test documents. When related requirement specification source documents changes, the test plan also needs to be updated with appropriate tests. When planning for regression tests, test plan is used to select test cases for execution by considering the requirement coverage.

1.7.3 TEST CASE

A test case normally consist of a unique identifier, requirement references from a design specification, preconditions, events, a series of steps to follow, input, output, expected result and actual result, test status, tester’s signature or timestamp.
1.7.4 TEST SCRIPT

Test script is a procedure or programming code or set of instructions that replicates user action or performed under test to check that the system works as expected. Test scripts are used in testing automation or automated testing.

1.7.5 TEST SUITE

Test suites are commonly known as collection of test cases. The test suite often contains more detailed instructions for each test case. A test case can be added to multiple test suites. It is created based on scope and can contain any type of tests.

1.7.6 TEST DATA OR FIXTURE

It is a data that is used in tests. All the test values and changeable environmental components are collected in separate files and stored as test data. Some test data is used to confirm the expected result, whereas some test data is used to verify the software behavior to an invalid input data.

1.7.7 TEST HARNESS

The software, tools, samples of data, input and output, configurations are all collectively referred as test harness. It enables the automation of tests. Test harness executes tests by a test library and generates a report. It requires that the test scripts are designed to handle different test scenarios and test data.

1.8 TYPES OF TESTING

Different types of software testing are carried out to accomplish different objectives when testing a software or application. Based on its objectives, testing can be organized and classified like functional testing, non-functional testing, Structural testing, manual testing, automated testing, unit testing, integration
testing, regression testing, acceptance testing, system testing, stress testing, security testing, performance testing, load testing etc.

1.9 CHALLENGES IN SOFTWARE TESTING

Both in manual or automation, Software Testing has lot of challenges. In general, the build is handed over to testers by the developers, assuming that the test team or testers will take lead on the build. This is happening in the organizations which does not follow the processes. Tester is the middle-man between developers and the customers, dealing with the pressure from both sides. The iron triangle is very common to many test managers. It is a model of control where cost, time and quality are shown on each corners of the triangle to express tension between the three objectives in product delivery. In product or system delivery, attempt at optimization of one these attribute will impact the others. Software Testing is seen as an expense in general, which is not the right attitude. This led to off-shoring where the cost of testers is lower. The disadvantage of this approach is that testers have been distanced from the stakeholders. This makes to offer quality testing a little bit harder. Test automation is another way that teams intent to reduce cost of testing, but it does not consider the hidden cost of maintenance which is needed to keep test scripts updated. Automation is an advantage but need to know where that is to be placed to make the testing really effective. By reducing waste, we can achieve the efficiency. There can be large amounts of wastage in testing that increases the cost and time of testing. The major challenges in testing are as listed below.

1.9.1 COMPLETE TESTING IS IMPOSSIBLE

There are various test combinations and scenario results. Testing all of them is an unrealistic goal. Clear focus on business goals and target users helps to prioritize the featured scenarios to test.
1.9.2 SETTING THE RIGHT PROCESS

The organization process for testing should be efficient in capturing the defects on time. Clear defined process helps to avoid misunderstandings. A uniform process has to be set up to make everyone on the same page. In real time it is difficult when teams are located in different time zones and locations.

1.9.3 RELATIONSHIP WITH DEVELOPERS

This requires skilled testers to deal with this relationship though they are finishing the work in their own way. There are hundreds of reasons testers or developers can make when they disagree with some points. To overcome this, testers also needs some trouble shooting skills and good communication skill.

1.9.4 LACK OF SKILLED TESTERS

This causes because of the wrong management decision when selecting testers for their project work. These less skilled testers may add more confusions than simplifying the testing phase. This results into insufficient, incomplete, and ad-hoc testing throughout the life cycle of testing.

1.9.5 UNDERSTANDING THE REQUIREMENTS

Often testers are responsible for communicating the understanding of requirements with customers. If tester fails to understand the requirements, the proper testing cannot be guaranteed. Testers require good understanding and logical thinking capabilities.

1.9.6 REUSE OF TEST SCRIPTS

The methods of application development are rapidly changing and it becomes difficult to manage the test scripts and test tools. Reusing test script or migrating test script is very essential but difficult to perform.
1.9.7 DECISION TO STOP TESTING

A very difficult decision to be taken is to decide when to stop testing. This requires deep knowledge of testing processes and importance of each other process and would also require on the fly decision ability.

1.9.8 TESTING ALWAYS UNDER TIME CONSTRAIN

When testers are provided with difficult time line to complete the testing, they simply focus on task completion and not on the quality of work or test coverage. There is huge list of tasks that needs to be completed within specified time which includes reviewing requirements, writing, executing, and automating the test cases.

1.9.9 WHICH TESTCASES TO EXECUTE FIRST

It is not possible to take the decision of which test cases should be executed and with what priority, if the testing has to be completed under time constrain. This requires good experience and understanding to assess which tests are important than others.

1.10 TEST SUITE MINIMIZATION

Test suite minimization is a vital test maintenance activity that tries to remove redundancy and minimize execution time and thus decrease the cost of testing. The test suites developed for testing can be reused and will be updated often as the software advances. Then new test cases will also be added to the test suite and the size of the test suite becomes outsized. Moreover, the test suite becomes superfluous and redundant as the requirements covered by some of the existing test cases will also be covered by new test cases as well. Thus executing or re-executing the outsized test suite consumes more time and also increases the testing effort. Therefore to minimize the effort and the time of testing, it is
essential to minimize the test suite and eliminate the redundant test cases from the original test suite. This process minimizes the number of test cases with respect to some coverage procedures, which is a set of rules that characterizes whether the software has been tested. Several minimization techniques have been proposed so far suffered by two major drawbacks: only single criterion based minimization is achieved and expected suboptimal solution is produced. A good minimization technique should allow to easily programming wide spectrum of test-suite minimization problems, handle issues that involve any number of criteria, and compute best solutions to minimize problems by leveraging modern integer linear programming solvers.

1.11 TEST CASE PRIORITIZATION

A method to prioritize and schedule the test cases available in a test suite is called Test case prioritization. The prioritization technique is enhanced in order to run test cases with higher priority, to minimize the execution time, effort and cost when doing software testing. Every organization has their own way and methods to prioritize or categorize the test cases. However, the one issue in test case prioritization techniques considers that there is only one test suite. A collection of a set of test cases is known as a test suite. There are no sufficient techniques available to resolve the issue of combination of more than one test suites. Prioritization can be done on the basis of requirements, history of the parent device, costs of bug fixing etc. Some of the factors that needs to be considered in prioritization are Mission-critical components, which are related to critical requirements, complex features, where failures would be most visible, areas of most frequent use, features that undergo frequent changes, areas with complex coding, areas with past histories of problems, new functionality, major functionality rather than going in detail.
1.12 MOTIVATION OF RESEARCH

Software test suite optimization is grabbing more attention due to many reasons. First, it reduces the execution time of a test suite to test the system in limited time. Secondly, to advance the turnaround time required between software enhancements during regression testing. Finally, the generation of effective test cases only makes high chances of finding out errors in the System. The literature study provides lot of ideas in the existing works on software test suite minimization problem. The observations on this study, has motivated this research work to propose a new framework for software test suite optimization. Few of the observations are given in this section. In general, the aim of test suite minimization is to minimize the length of a test suite and at the same time preserving its effectiveness as well.

An optimization algorithm based on Bacteriologic algorithm has been applied on various test benches. The further generation of test cases loses the advantage of combining the best properties of both the parents since BA doesn’t have cross over operator (Benoit Baudry et al, 2002).

The existing tools cannot be tested in model based test data generation for tools testing, since it requires instrumentation in the software, which needs association with the tool vendor. The basis models are represented in graphical image when using third party created code generators, which cannot be identified by this model (Ingo Stuermer et al, 2007).

There is no idea or feature given for software testing optimization in Fuzzy Logic based test data generation. There is no method has given for assessing the probability which is used to deal uncertainty (Zhang L et al, 2009).

In the proposed Search based approach, huge amount of memory is needed in terms of long term memory to overcome of stuck up at local optima and short
term memory to memorize all the test cases assigned for the current search. Though this short term memory is removed often, this will cause a vital problem of effective memory utilization (Diaz J.C et al, 2008).

The above observations have encouraged this research work to develop a new test suite optimization framework which encompasses of the benefits of all of the discussed methods and at the same time reduces the drawbacks of them. The proposed framework is a hybrid optimization framework which is a combination of test suite minimization and test case prioritization and achieving the optimization using Gravitational Bee colony algorithm with data featured fuzzy logic.

1.13 RESEARCH CONTRIBUTIONS

The major contributions of this research work includes the design and implementation of a new two level hybrid test suite optimization which is mainly focusing on both test suite minimization and test case prioritization. The proposed method (GBC with Fuzzy) is having the first level hybridization of two population based heuristic algorithm known as Artificial Bee colony algorithm and Gravitational Bee search Algorithm, which gives a proposed method GBC for test suite minimization and then having the second level hybridization of data featured fuzzy logic with GBC for test case prioritization. The proposed method gives promising results on various efficiency aspects of a test suite.

All the information technology and software organizations have the challenge of reducing the time and effort spent on software testing. The proposed framework will serve the purpose and achieve that optimization aim.
1.14 PROBLEM STATEMENT

The test suites which are used for testing are often reused and will be appended with new test cases frequently, whenever the software get enhanced. When new test cases are added to the test suite, the size of the test suite in terms of its length increases. Also there are possibilities for the test suite to become outsized and redundant. Very often the new test cases getting added for the requirements, which are already met by other existing test cases. Thus test execution with such redundant and outsized test suite takes more time and increases the overall testing effort and time. Therefore to minimize the effort and the time of testing, it is necessary to eliminate the redundant test cases and minimize the test suite. Thus, the focus of this research work is to minimize the test suite, by finding a sub set of test cases that meets the same or better coverage as the original test suite, based on some criteria rules. By reducing the number of test cases in the original test suite, the testing efforts can be reduced and in turn the time of Software testing will also be reduced. Thus the objective of this research is minimizing the ‘to be executed’ test suites, which will reduce the effort and execution time of testing. And prioritizing the test cases to further provide better fault detection and coverage. Most of the research on optimization technique for test suites compares only two or three parameters in comparison analysis. As part of this research work, five parameters have been compared to check the efficiency of the proposed frame work. Most of the research works focused on either test suite minimization or test case prioritization. The proposed framework extracts the benefits of both techniques and achieves better efficiency than few of existing optimization techniques.

To obtain the optimized test suite, the first step is to find a minimized test suite which is a subset of original test suite the meets the same requirements met by the original unreduced test suite. Then this minimized test suite is prioritized based on the fault detection and criticality. This yields us an optimized test suite
which helps to achieve the testing with less effort and execution time. The technical problem statement of this work is as follows.

Given a set of test cases \( T \) is defined as \( \{t_1, t_2, t_3, \ldots, t_n\} \), a set of defined testing requirements is defined as \( R \{r_1, r_2, r_3, \ldots, r_m\} \) that must be covered to given code (program), trained by each and every test case in \( T \). The minimizing test suite problem is to find a subset \( T_1 \) of \( T \) and it must be in minimal cardinality that meets the same set of requirement as those trained by the un-reduced test suite \( T \). \( PT \) is the set of permutations of the minimized test suite \( T_1 \), \( f \) a function from \( PT \) to the real number. Find \( T_1' \in PT \), such that (for all \( T_1'' \)), \( T_1'' \in PT \), \( (T_1'' \neq T_1') \), \( f(T_1') \geq f(T_1'') \).

\section*{1.15 METHODOLOGY}

There are two main phases in the proposed GBC with Fuzzy framework. We generate the initial test cases using one of the existing test generator tool called Breakpoints Matrix (Auto Test generator) for the given test requirements from Software Requirement Specification document. After all of that, as shown in Figure 1.4, the initial test suite is the unreduced test suite that we are going to minimize and optimize with our proposed framework. The multi objectives criteria are defined as minimize the size of the test suite that is number of test cases and coverage of the test suite.

To minimize the test suite, the new hybrid Gravitational Bee Colony search algorithm (GBC) which is the combination of Artificial Bee Colony optimization and Gravitational Search Algorithm is used. This proposed work is to minimize the test suite and time by discovering a set of test cases that gives the same coverage as the original unreduced test suite based on some criterion which is done based on concepts of GSA and ABC.
1.15.1 ARTIFICIAL BEE COLONY (ABC) ALGORITHM:

ABC algorithm is a popular swarm based Meta heuristic algorithm which imitates the real honey bees searching method. ABC is a well-known optimization method which is based on population search, where the bees modify the position of the food. Locating the position of food with high nectar volume is the major goal of bees. ABC algorithm contains three types of bees such as onlooker bees, employee bees and scout bees. Based on employee bee’s dance, the onlooker bees choose their food source. The employee bees goes to look for their food source and return back to the hive and perform a dance on this region. When employee bees find more abandoned food source, they becomes scout bee and again look for a new food source. The nectar source is chosen using a nest mate in sequence who found the food source previously. The bees dance in the “hive”, on nectar’s sources which are used to convince other bees to follow them.

1.15.2 GRAVITATIONAL SEARCH ALGORITHM (GSA):

GSA is a newly materialized population based stochastic optimization approach which is based on mass interaction and gravity. This method presents
an approach that navigates over a multi-dimensional search and imitates the mass interactions. In GSA, the particles are considered with their performances which are measured by their masses. In this process every particle is defined as a nominee solution to the given problem.

Particles are attracting each other by their gravity power, and this force can make a global movement of all the particles with higher masses. Therefore, masses in general, works with each other by a direct form of interaction with gravitational force. Subsequently, the higher masses have greater fitness values, indicates the good solution to the given test suite searching problem. Providing the target function produces better results, here the search agents defined as a collection of bodies.

1.15.3 GRAVITATIONAL BEE COLONY (GBC) ALGORITHM:

In this research we propose a new hybrid test case selection method which is the combination of Gravitational Search Algorithm (GSA) and Artificial Bee Colony (ABC) algorithm. This is named as Gravitational Bee Colony (GBC). This proposed work is to reduce the test suite and time by finding a set of test cases that meet the same coverage or better than the original test suite based on some criterion based on GSA and ABC. The proposed technique uses the initial test suites generated by Breakpoints Matrix (Auto Test generator) tool and it also covers the faults that are detected already.

Bees are utilized as agents; one part of the bees will start foraging on randomly chosen test cases. After this process, the new test cases will be added to the discovered path by the bees and that increases fault detection capacity. After adding a new test case, the bees come back to their hive, and exchange the obtained information by using GSA. The Gravitation law is used to process the exchanged information. The distance between the two bees is calculated by Motion Law with the help of bee’s current and previous velocity. The current
velocity of the bee is equivalent to the bee’s sum of coefficient previous velocity. The new set of test cases are generated after the gravitational law and motion law and utilized to search by new bees. This procedure repeats until any of the bees has find a new set of test cases which covers additional faults. The bees start performing waggle dance once they identified such new test case.

1.15.4 GRAVITATIONAL BEE COLONY WITH FUZZY (GBC with Fuzzy)

After identifying the objective functions of test suites, the prioritization is done by data featured Fuzzy logic for efficient test suite execution. The clustering concepts of data mining are used to traverse through the minimized test suite and three weights related to the prioritization are defined as high, medium, low where the test cases nominated by GBC are relevant, partially relevant or non-relevant to the requirements. However, the weight can be considered based on the Gravitational Force.

1.16 OVERVIEW OF RESEARCH

Software Testing is one of the essential and expensive activities in Software development lifecycle due to its effort and time required. One of the key focus in software development process is to reduce the effort of software testing and cut down the time of testing. The test suites developed for testing can be reused and will be updated often as the software advances. This make the test suites as superfluous and redundant, as the requirements covered by some of the existing test cases might also covered by new test cases as well. Thus executing or re-executing the outsized test suite consumes more time and also increases the testing effort. Therefore to minimize the effort and the time of testing, it is essential to minimize the test suite and eliminate the redundant test cases from the enhanced test suite. Thus, the focus of this research is to minimize the test suite by discovering set of test cases that gives the same coverage as the original test suite based on some criterion.
This thesis is organized into eight main chapters as follows. Each chapter has written to be largely self-contained and complete. Chapter 2 provides brief discussion on literature survey and Chapter 3 discusses about a detailed study on various testing methods available. Chapter 4 briefs about a novel method for reduction of regression testing efforts. This method applies an existing minimization algorithm to minimize the regression test suite and the selection of regression test cases is achieved by the proposed methodology and also provides the analysis and result summary of the work. Chapter 5 explains about a modified genetic approach which is proposed to reduce the regression test efforts. It also discusses about the analysis and results of this approach. Chapter 6 discusses about a detailed study on major test suite optimization methods available. Chapter 7 describes a hybrid test suite minimization approach which is a combination of Particle swarm optimization and Ant colony optimization. This section also provides the analysis and results of this work. Chapter 8 discusses about the new proposed Gravitational Bee Colony with data featured Fuzzy logic optimization technique for efficient test suite minimization and test case prioritization. This section also provides clear view of this proposed methodology, analysis and respective results and a detailed conclusion of this work. Chapter 9 discusses about the Test suite minimization based on the new proposed method GBC with Fuzzy and compares it with other two optimization techniques and provide the compare analysis and its results. This also clearly shows the efficiency of the proposed approach with the conclusion. Chapter 10 describes the overall analysis and comparison of results of proposed method with original unreduced test suite, and the two other optimization method. Chapter 11 conclude this research work with an overall conclusion of the efficiency of the proposed GBC with Fuzzy framework on test suite optimization. It also summarizes about the achievements of this research work and future opportunities and thought processes to bring initiative further.