MULTI-OBJECTIVE REGRESSİON TEST-SELECTION IN PRACTICE: AN INDUSTRIAL CASE STUDY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF

MIDDLE EAST TECHNICAL UNIVERSITY

BY

RAMAZAN ÖZKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JANUARY 2017

MULTI-OBJECTIVE REGRESSİON TEST-SELECTION IN PRACTICE:
AN INDUSTRIAL CASE STUDY

Submitted by Ramazan Özkan in partial fulfillment of the requirements for the degree of **Master of Science in The Department of Information Systems Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Director, Graduate School of Informatics _____

Prof. Dr. Yasemin Yardımcı Çetin
Head of Department, Information Systems _____

Assoc.Prof. Dr. Aysu Betin Can
Supervisor, Information Systems _____

Assoc.Prof. Dr. Vahid Garousi
Co-Supervisor, Hacettepe University _____

Examining Committee Members:

Prof. Dr. Onur Demirörs
Information Systems, Middle East Technical University _____

Assoc. Prof. Dr. Aysu Betin Can
Information Systems, Middle East Technical University _____

Prof. Dr. Ali H. Doğru
Computer Engineering, Middle East Technical University _____

Assoc. Prof. Dr. Altan Koçyiğit
Information Systems, Middle East Technical University _____

Assist. Prof. Dr. Ayça Tarhan
Computer Engineering, Hacettepe University _____

Date: 12.01.2017

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : **Ramazan Özkan**

Signature : _____

ABSTRACT

MULTI-OBJECTIVE REGRESSİON TEST-SELECTİON IN PRACTICE:
AN INDUSTRIAL CASE STUDY

Özkan, Ramazan

M.Sc., Department of Information Systems
Supervisor: Assoc. Prof. Dr. Aysu Betin Can
Co-Supervisor: Assoc. Prof. Dr. Vahid Garousi

January 2017, 90 pages

Regression testing is testing of previously verified parts of a software system to make sure that software changes do not affect those parts. However running an entire regression test suite after every code change may be costly or even infeasible due to time and resource constraints especially in large-scale software projects. In order to resolve this issue and optimize the number of test cases to be rerun for regression, several techniques have been proposed in the literature. One of these is Multi-Objective Regression Test Optimization (MORTO) approach.

This thesis is an "action research"-based empirical study to improve regression test-selection process of an industrial project, in the defence sector, based on the MORTO approach, in which the problem is formulated and solved by using a genetic algorithm. The empirical results demonstrate that this approach yields a more efficient test suite based on a set of five cost objectives, four value objectives and a dependency while providing full requirements coverage.

Keywords: Regression selection, multi-objective optimization, genetic algorithm, empirical study, industrial case study

ÖZ

ÇOK-OBJEKTİFLİ REGRESYON TEST SEÇİMİ UYGULAMASI:
BIR ENDÜSTRİYEL VAKA ÇALIŞMASI

Özkan, Ramazan

Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Doç. Dr. Aysu Betin Can
Ortak Tez Yöneticisi: Doç. Dr. Vahid Garousi

Ocak 2017, 90 sayfa

Regresyon testi; yazılım üzerinde yapılan değişikliklerin, yazılımın daha önce çalıştığı doğrulanmış bölümlerine etkisinin olup olmadığının tespiti amacıyla yapılan testlerdir. Ancak yapılan herbir değişiklik sonrasında bütün testlerin tekrar edilmesi hem maliyet etkin değildir hem de özellikle büyük boyutlu projelerde zaman ve kaynak limitlerinden dolayı uygulanamamaktadır. Bu problemin çözülebilmesi ve tekrar edilecek test miktarının optimizasyonu için literatürde bir çok yöntem önerilmiştir. Bu yöntemlerden biri çok amaçlı regresyon test seçimi yaklaşımıdır.

Bu tez endüstriyel bir projeye ait regresyon test seçim işleminin çok amaçlı regresyon test seçimi yaklaşımı kullanılarak geliştirilmeye çalışıldığı deneysel bir çalışmadır. Bu çalışmada çok amaçlı regresyon test seçimi yaklaşımı genetik algoritma kullanılarak formüle edilmiş ve çözümlenmiştir. Deneysel çalışma sonuçları çok amaçlı regresyon test seçimi yaklaşımının değişiklerden etkilenen isterlerin tamamını kapsayan ve aynı zamanda on farklı maliyet ve değer kriterine gore optimize edilmiş bir çözüm sağladığını göstermektedir.

Anahtar Kelimeler: Regresyon seçimi, çok-objektifli optimizasyon, genetik algoritma, deneysel çalışma, endüstriyel vaka çalışma

dedicated to *my wife R.Aytül ÖZKAN,*

my son Akın Alp ÖZKAN and

my doughter who is not born yet

ACKNOWLEDGMENTS

Many thanks to my supervisor, Assoc. Prof. Dr. Aysu Betin Can and my co-supervisor, Assoc. Prof. Dr. Vahid Garousi, for their invaluable guidance and constructive criticisms throughout the development of this thesis. It has been a great privilege to have been mentored by such esteemed researchers.

Special thanks to my colleagues who provided encouragement and support for this study. This thesis would not have been written without them.

And I would like to thank my little son for his great effort to keep me periodically away from study which helps me to refocus.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

CHAPTER 1

INTRODUCTION

1.1 Motivation/Rationale

Although a given software system may have been tested during its development to satisfy a set of given adequacy criterion, it usually evolves with bug fixing, optimization, enhancement or adaptation activities in time and thus has to be retested during maintenance and evolution phases. Basic reasons of software maintenance and evolution are changing user needs or environment, detected errors, stability issues and performance issues. Regression testing is an activity which is carried out to make sure that this evolution does not affect the approved functionality of the software system [1-3].

The traditional and simplest regression testing approach is to rerun all the test cases that have been used to verify the functionalities of the software before the modifications have been made, which is called the "retest all" technique [2]. In practice, repeating all test cases which are previously executed successfully after each software revision is not practical due to time and budget constraints especially in the case of large-scale software systems.

An alternative approach is selecting only a subset of the initial test suite called regression test-selection. The regression test-selection process also has substantial costs and may reduce fault detection effectiveness. This balance between the cost required for selecting and rerunning and the fault detection effectiveness of the selected test cases is the subject of regression test-selection [4]. To find cost-efficient regression test selection methods in different aspects of regression testing, a large amount of research effort has been spent and several techniques have been reported in the literature. Also there are analytical, comparative and empirical evaluations of individual techniques [2, 4-9] in the literature.

Moreover, until 2007 all of the studies using different selection techniques are structured on single objective approach. After the first multi-objective example [10], most of the regression test-selection studies use this approach; but these studies are structured on two or three objectives. This limitation adversely affects the effectiveness of regression test-selection.

This thesis is an action research [11, 12] type empirical evaluation of multi-objective regression test-selection approach on an industrial software project and it aims to improve manual regression test-selection process of the subject project. It is structured as a customized version of Multi-objective Regression Test Optimization (MORTO) approach defined by Harman [13]. The cost and benefit objectives, as proposed in the original MORTO paper [13], have been selected and some

have been extended based on the collected data from the subject software system. For realization of MORTO approach, a genetic algorithm has been developed.

1.2 Scope and Goals

In the subject project, regression test-selection process has been performed manually in the past and this has made the cost and quality of the testing highly dependent on the skills and experience of the test team. For customer side, fundamental determiner in this process is coverage of the affected requirements. On the other hand, contractor mainly focuses on cost of the regression test suites. There has been a need for systematic and cost efficient approaches to regression test selection. Based on these needs, the main goal of this thesis is to provide full coverage of affected requirements and to improve the overall cost-benefit of regression testing approach based on all the relevant regression selection criteria in the project.

The subject project is an ongoing project in acceptance phase in which there is no access to source code. Because of the current phase of project and source code access limitation, only black-box testing that can be run on integrated system is analyzed in this work. White box tests are not in the scope of this thesis. Test cases that tests only functionality of the system are subject of this thesis. Nonfunctional test cases such as performance, stability, latency are out of scope of this thesis. These test cases should be stated in each regression test suites.

Based on a careful literature review, Multi-objective Regression Test Optimization (MORTO) [13] has been selected as the most suitable approach in this thesis. Main reason of this selection is having multiple criteria that effects regression test selection. Solution is customized to meet the requisites of subject problem.

In order to show the effectiveness of Multi-objective Regression Test Optimization approach, regression test suites selected by subject approach are compared with the test suites selected by manual regression test-selection process currently used in subject project and test suites selected by the selective coverage based regression test selection method proposed by Gu et al. [14]. Multi-objective Regression Test Optimization and existing regression test selection approach comparison is done in test suite size, coverage of the requirements that are affected and not affected by the code modification (the unaffected requirements are called “irrelevant” requirements in the literature), and fitness value (cost-benefit balance) aspects for five different software versions of the project. On the other hand, Multi-objective Regression Test Optimization and selective coverage based regression test selection comparison is done in test suite size, coverage of the requirements that are affected and not affected by the code modification, fitness value (cost-benefit balance) and execution time aspects for same five different versions of the project.

1.3 Contributions

Based on real industrial needs, this thesis has tackled a real life regression selection problem with several conflicting objectives. The approach can be applied to other similar contexts. Below are the primary contributions of this thesis:

- A multi-objective formulation of the regression test selection problem which is a customized version of multi-objective regression test optimization proposed by Harman [13] is introduced. It combines different cost and value objectives.

- It presents a genetic algorithm for solving multi-objective regression test-selection problem. In order to provide full coverage of the requirements that are affected by the change in the system under test (SUT) which is one of the objectives, a constraint function is applied to each step of genetic algorithm that creates solution.

- It provides empirically evaluation of tailored multi-objective regression test optimization on an industrial project with specifically optimized genetic algorithm. This optimization covers population size parameter and variation operators which are crossover rate and mutation rate parameters of genetic algorithm.

- It also presents the results of empirical evaluation including comparison of proposed regression test-selection process with manual regression test-selection process used in project and selective coverage based regression selection method [14] in test suite size, coverage of the requirements that are affected by the change in the SUT, coverage of the requirements that are not affected by the change in the SUT and fitness value (cost-benefit balance) perspective. This comparison shows that tailored multi-objective regression test-selection approach is capable of providing a more efficient regression test suite in test suite size, affected requirement coverage and fitness value perspective.

1.4 Definition of Terms

In this section some of the most commonly used terms related to regression test-selection are defined.

Term	Definition
Acceptance Test	It is a test or test collection that is used to determine if the requirements of a software contract are met.
Affected Requirements and Irrelevant Requirements	In regression test-selection concept "Affected Requirements" statement means requirements which have relation with software modifications from test suite-covered requirements set. Contrary, "irrelevant requirements" statement means requirements which have not a known relation with software modifications.
Change Impact Analysis	It is the process of finding potential effects of a software change on existing functionalities.
Heuristic	It is a kind of problem solving approach which does not guarantee the perfect solution, but sufficient for the immediate goals. It is used when finding best solution is impractical.

Term	Definition
Regression Test Suite	It is a test suite that is used to show that modified parts of software do not affect previously verified part of the software
Requirements	It is a condition or capability defining a customer need in a contract or other formal document that must be met by a software system. High level requirement defines a specific functionality of a system. On the other hand low level requirement defines design decisions to implement that functionality.
Requirement Coverage	It is a measure of the amount of requirements that are covered. It means proportion of requirements that are tested by a specific test case or test suite to total number of requirements.
Test Case	It is a scenario that executes a set of functionalities of a software system to determine whether these functionalities are working as defined in requirements.
Test Suite	It is a set of test cases that are used to test software to make sure that it has the specified behaviors.

Table 1-1 Definition of terms

1.5 Organization of This Thesis

The thesis is organized as follows: Section 2 explains the basic background information of important concepts used in this thesis and overviews the related studies in literature according to underlying goals of techniques that give direction to selection process and criteria used for selection are reviewed. Section 3 explains the research methodology and gives the case description and needs for the study. Section 4 describes the genetic algorithm structure to be able to realize MORTO. Section 5 explains the application of MORTO to an industrial project and demonstrates the effectiveness by analyzing results of existing regression test-selection process and MORTO. Finally, a summary of the conclusions is stated in Section 6.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Background

In this section basic background information of important concepts frequently referred to in this thesis is provided. These are empirical study, action research, regression testing, regression test-selection as an optimization problem, solution methods for multi-objective optimization problems, and genetic algorithm as optimization solution algorithm. Since these important concepts may be interpreted differently in different contexts their interpretation within this thesis is clarified as they are introduced here.

2.1.1 Empirical Research

Empirical research tries to find out, estimate or explain facts in different aspects based on observed or experienced evidences. These evidences are obtained and evaluated by experimentation, systematic observation, surveys or interviews, or by the careful investigations of documents or artifacts.

Data collection and analysis can be conducting by using both qualitative and quantitative methods in empirical research approaches. Qualitative methods collect material in the form of text, images or sounds obtained from observations, interviews and documentary evidence, and analyze it by using methods that do not rely on precise measurement to product their conclusions, while quantitative methods collect numerical data and analyze it by using statistical methods [16].

There are many empirical research approaches in literature but except Action Research, describing empirical methods and application steps is beyond the scope of this thesis. However, it can be said that all empirical methods basically require five steps:

- Specify a research question,
- Propose a solution,
- Collect data,
- Analyze the data, and
- Interpret the data.

Action Research

Action research is a problem solving research by which researchers and practitioners attempt to study real life problems in a scientific concept. The concept of action research includes trying to resolve an existing practical problem and observing if this effort has been fruitful; if it was not, then trying again to resolve the same problem [16]. Different than the most empirical research methods, action researchers aim to intervene in the studied situations with the purpose of improving the situation. It provides practitioners the opportunity to study and improve their own practice and researchers to understand the limits and applicability of developed principles, tools and methodologies.

In order to explain the fundamental features of action research, a comparison between action research and basic research is given in Table 3-1.

Basic Research	Action Research
Develop theories and make generalizations	Primarily focused on a particular issue or concern in a single organization
Conducted by researchers	Conducted by researchers and practitioners
Conducted in controllable environment	Conducted in real environment
Creates conceptual solutions for conceptual problems	Creates solutions for real world problems and proposed solutions are used in real world

Table 2-1 Comparison of basic research and action research[17]

Action research combines theory and practice with as much participation of practitioners as researchers to each phase of study that requires decision making. This feature creates synergy between the practitioners and researchers to apply ideas to real world problems. In this strategy, action means a kind of transformation in a community, organization or project, while the research means understanding of subject transformation phenomenon by all participants of the study. There are two forms of action research strategy. First one primarily focuses on action and research is by-product while other primarily focuses on research. For thesis purposes I choose the form that primarily focuses on research.

Because of the direct relation with practice, this thesis is conducted as an action research. Following section explains the general steps of an action research.

Conducting Action Research

Conducting an action research includes four basic steps: (1) determine an issue or topic to search, (2) collect required data related to issue determined in first step, (3) analyze, test and evaluate the collected data, and (4) execute action planning, which means the application of the action research results [16].

Identify the Topic or Issue

In this step topic or issue of action research is determined. It generally includes a pressing problem in a real environment or a promising practice for a real environment. In this thesis the topic is improving manual regression test-selection process of an industrial project. After determination of topic, a literature review is performed and research questions are determined in this step. This step of action research is handled in chapters 1, 2 and 3 of this thesis.

Collect the Data

Intrinsically many data sources are suitable for action research. It can be quantitative or qualitative like demographic information, test results, observations, or questionnaires. In this thesis test record sheets, test procedures, software version description documents, test-requirement compliance matrix, regression test plans and test results, interviews with test team and decisions taken by test team in meetings are used as data sources. This step is presented in Chapter 4.

Analyze the Data

Primary intent of action research is to use collected data to improve practice. This step narrows collected data to categories or features in order to provide usable information to next step of action research. This step is presented in Chapter 5.

Carry out Action Planning and Share the Findings

Final step of action research is to compare old and new version of practice and show improvement. This step is presented in Chapter 5.

2.1.2 Regression Testing

Subject project of this thesis is an ongoing project in acceptance phase. Test case list of project is previously determined and the aim of acceptance phase is to run all of the test cases successfully at least one time. A new version of system software is launched after resolution of detected faults in previous version. In order to verify resolutions, failed test cases are repeated as retest. On the other hand, in order to verify modifications to be done for resolution of detected faults have not caused unintended effects a subset of previously verified test cases which is named as regression test suite is run. And this cycle continues until successfully running of all test cases at least one time.

In this perspective, regression testing is repeating a subset of test cases that are previously verified based on impact analysis.

2.1.3 Regression Test-Selection: As a Multi-objective Optimization Problem

Optimization can be described as finding the best solution from a set of available alternatives based on determined criteria. According to the number of criteria, optimization process can be classified as single objective optimization or multi-objective optimization. In single objective optimization, selection is done based on only one criterion and it does not need user decision. On the other hand, multi-objective optimization tries to balance many and possibly conflicting criteria and it needs user decision to prioritize criteria.

Regression test-selection process try to reduce the cost of test suite to be executed while satisfying the testing criteria. Most of the regression test-selection related studies focus on code coverage criterion but there is not a certain correlation between code coverage and fault detection [10]. Naturally, there are lots of criteria that can affect the regression test-selection process and generally these criteria are conflicting factors such as execution time and requirement coverage.

In this perspective regression test-selection can be interpret as a multi-objective optimization that is capable of taking into account many conflicting objectives and tries to find the most suitable test suite option that satisfies determined criteria. This approach is firstly proposed in the study of Yoo and Harman [10] with two and three objectives. Then it is extended in the study of Harman [13] which is used as a baseline for this thesis. That study emphasizes necessity of multi-objective approach for regression selection and provides a list of possible regression test-selection objectives. Extended or direct versions of applicable ones from that list are used in this thesis.

2.1.4 Solution Methods for Multi-objective Optimization Problems

In multi-objective optimization problems, finding a point that simultaneously optimizes all objectives of a problem is generally not possible. Therefore, different techniques are proposed in literature to solve multi-objective optimization problem. These techniques can be grouped under two fundamental approaches: Pareto (Posteriori) Approach and Scalarization (Priori) Approach.

Pareto Approach

In Pareto Approach, a group of Pareto optimal solutions are found then one of these solutions is selected based on preferences. Each solution includes a value for each objective function and Pareto optimal solution means a point in which at least one of the objective functions has lower value than other solutions. A graphical demonstration of Pareto approach for a two-objective problem which includes Pareto points and Pareto curve is stated in Figure 3-1.

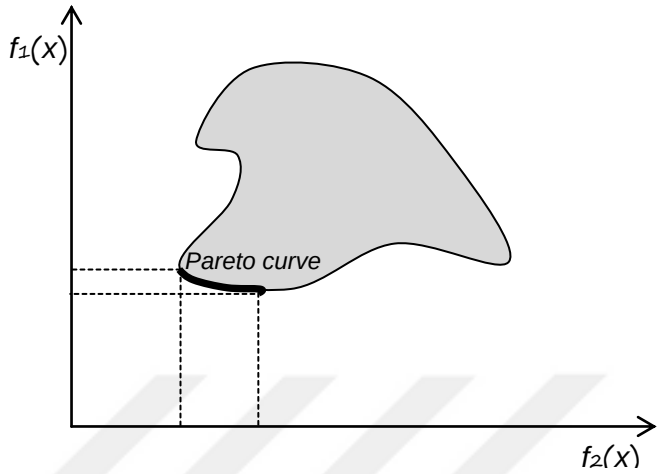

Figure 2-1 Example of Pareto Approach

Computing all Pareto points cannot be accomplished efficiently in many cases especially for the cases that include more than three objectives. Theoretically, finding all Pareto points exactly is possible, but this process needs exponential size computation power.

Scalarization Approach

Because of the drawbacks of Pareto approach, a multi-objective problem is often solved by combining its multiple objectives into one single-objective scalar function. This approach is in general known as the weighted-sum or scalarization method. In this approach decision maker chooses appropriate weights for each objectives and multi-objective optimization problem is transformed to a single objective optimization problem by using these weights. After that a solution satisfying these preferences is found as optimization process. Scalarization approach is used for solution of multi-objective regression test-selection problem defined in this thesis.

2.1.5 Optimization Algorithms

There are multiple search based optimization algorithms defined in literature for multi-objective optimization problems. Genetic algorithm is one of them and it is used in solution part of this thesis. Because of that only genetic algorithm is explained in this section. Other search algorithms are out of scope of this thesis.

Genetic Algorithm

Genetic algorithm (GA) is a heuristic search based optimization algorithm that imitates the natural selection process [18]. It is generally used to find useful solutions for optimization and search problems. The main goal of algorithm is evolving a population of possible solutions toward better solutions.

GA starts with the creation of a population named as initial population from randomly generated individuals. In each iteration, a new population called a generation is generated by using selection and variation (crossover and mutation) processes of GA. For the selection process of GA,

the fitness value of each individual in the population is calculated and individuals that have better (based on the problem, better means less or more) fitness value are selected from the population. Then variation processes which are crossover and mutation are applied to individuals selected in previous step to form a new generation. This process continues until algorithm reaches its termination criteria which can be a maximum number of generations that are produced, or a satisfactory fitness level for the population. A basic flow diagram of classical genetic algorithm is given in Figure 3-2.

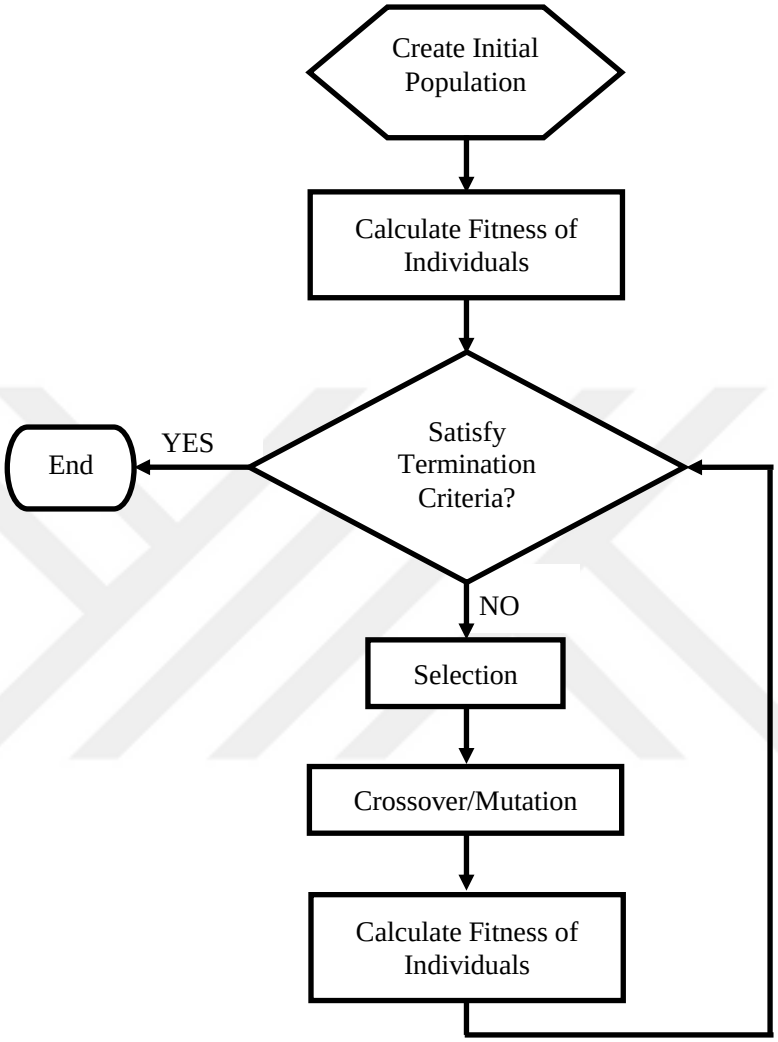

Figure 2-2 Genetic Algorithm Flow Diagram [19]

A genetic representation of the solution domain and a fitness function to evaluate it are required to apply genetic algorithm to a specific problem. After creation of the genetic representation and fitness function, a GA starts to initialize a population of solutions and then to improve it with selection, crossover and mutation processes in each iteration.

2.1.6 Data Collection

Data is a collection of numbers and characters which correspond to quantitative or qualitative values of variables. Quantitative data is related with quantities and it is written down with numbers. The main feature of quantitative data is its measurability like height or shoe size. On the other hand, qualitative data is related with qualities of variables which are immeasurable like softness of something.

Data collection is the process of gathering these values and measuring them in order to able to answer research question. In data level perspective, mainly there are four types of data levels: ordinal level data, nominal level data, ratio level data and internal level data. Nominal level data is the data which is used as labels for variables. It is not usable in arithmetic processes. For example; in this research, requirements' codes and test case cost subtypes codes are nominal level data. Ordinal level data represents a meaningful order for values of a variable. For example, test case execution priority which means the preference of test cases is an ordinal data. It gives the order of values but it does not provide the distance or ratio between values. Internal level data are used to measure any attitude that is gradually ordered. Internal level data is similar to ordinal level but the main difference is that the distance between values of internal level data is same. Ratio level data includes the pure numerical data that can be used in arithmetic processes like sales of a company or expenditure of a company.

Delphi Method

There are many data collection methods. Details of these methods are not subject of this thesis. However, Delphi Method which is used in this thesis for data collection is explained in this section.

The Delphi method is a useful way to collect data with consensus of a group of experts. It is a kind of structured communication technique relying on a panel of experts. In this method, experts answer the questionnaires in two or more rounds. Each round, responsible or coordinator person gives a brief summary of current status to provide an environment suitable for convergence to reach a consensus. In empirical study part of this thesis, some of the data is collected by using Delphi method. They are explained in Chapter 5.

2.2 Related Work

In this section, existing regression test-selection techniques in literature are scrutinized in two different approaches. In first part, underlying goals of techniques that give direction to selection process are reviewed. In second part, criteria that are used for selection are reviewed.

2.2.1 An Overview of Regression Test-Selection Techniques

Several regression test-selection techniques have been proposed in the literature [4]. Because of the differences in underlying goals, regression test-selection techniques produce clearly different results in test selection process. Based on these differences, regression test-selection techniques can be group in three different families [8, 20-22]:

Modification-Based Techniques (Selection)

Modification-based techniques select regression test cases based on different outputs produced by old and new versions of program. In these techniques, tests that will cause the modified program to produce different output than the original program are selected.

Most of the techniques focused on software modifications in subject systems by using source code analysis type methods such as execution trace analysis, data flow analysis and control flow analysis [1, 3, 23-29]. These Regression test-selection methods do not provide a specific structure to be easily applied in another context by software developers and they require quite much effort. If the cost of test selection, execution of selected test cases, and validation of the results is less than

rerunning all test cases, test selection will be practical [4]. This concern is valid especially for large-scale and complex systems that require analysis of interaction between several components and layers. A high-level analysis that correlates code units to test units is more useful for large-scale and complex systems. Besides; source code access may not be possible for all cases and source code analysis based techniques could not be used for those cases.

Code Analysis-Based Techniques

Code Analysis based regression test selection techniques aim to find test cases that produce different output for old and new versions of the program by using source code analysis. These techniques require deep and time consuming static analysis in source code to map changes to tests and they are not applicable to large-scale and complex systems. A great number of code analysis based techniques are proposed in the literature. However, limited number of empirical evaluations of regression test-selection techniques are carried out in a real industrial context [8]. Most of the studies employing code analysis in the literature are structured based on below methods.

- ***Data Flow Analysis***

In this technique, test case selection is done based on the examination of data interactions that have been affected by modifications. To find these test cases, definition-use pairs of the variables are analyzed and test cases executing the path from definition to use of the modified variables are selected. An example of dataflow coverage-based regression test-selection technique have proposed by Harold and Sofa [30].

- ***Control Flow Analysis***

These kinds of techniques are structured based on the analysis of Control Flow Graphs (CFG) differences of original and modified program. CFGs for both versions of the program are constructed and execution trace of each test case on these CFGs are recorded and compared. Rothermel and Harold [3] have proposed a control flow analysis technique named as Graph Walk. In this technique, control flow graphs of old and modified program are compared and if any node in control flow graphs of old program is not equivalent to corresponding node in modified program, all test cases that execute mismatching node are added to test suite.

- ***Execution Trace Analysis***

The execution trace of a test case on a program means the execution sequence of program statements that are executed with the test case. In this technique execution traces of test cases for old and new versions of the program are compared and test cases have different execution paths are selected. Orso et al. [31] and Akhin and Itsykson [29] used this technique in their studies. Vokolos and Frankl [32] proposed a tool named as Pythia. That tool uses a slightly different version of execution trace analysis. It keeps a history of the basic blocks executed by each test case and to identify the modified program statements, compares the source files of the old and new versions of the program.

Requirement Analysis-Based Techniques

Requirement based techniques aim to find an optimum test set that provides maximum coverage of the affected test requirements with minimum coverage of the irrelevant ones. Affected test requirements mean the requirement group that are affected from modifications, while irrelevant requirements mean ones that are not affected from modifications. Different versions of requirement

coverage based regression test-selection method are used in literature. Chittimalli and Harrold [33] proposed the basic requirement coverage based regression test-selection method in which regression test suite is created by including modification related requirements. Krishnamoorthi and Mary [34], Srikanth et al. [35] and Gu et al. [14] improved that method by using additional factors other than requirement coverage such as irrelevant requirement coverage, customer priority, fault impact and implementation complexity.

For example, selective requirement coverage based regression test selection technique proposed by Gu et al. [14] is structured on maximizing coverage of affected requirements, while minimizing coverage of irrelevant requirements based on a weighting factor. Given the weighting factor α, for a single test case, its fitness criterion (denoted by F_j), is defined as weighted average of its contribution criterion and futile criterion υ_j.

$$F_j = \alpha \cdot \varsigma_j + (1 - \alpha) \cdot \upsilon_j$$

Contribution criterion of a test case is the proportion of affected requirements covered by test case under analysis to the total number of affected requirements. On the other hand futile criterion is one minus the proportion of irrelevant requirements covered by subject test case to the total number of irrelevant requirements.

Model-Based Techniques

Model based Regression test-selection techniques are structured based on the design models like sequence diagrams, case diagrams or class diagrams [36]. Changes on these models and their impacts on previously verified test suite are analyzed. For example, UML based selective regression testing strategy proposed by Farooq et al. [37] uses state machines and class diagrams for change identification. Another study uses model based approach was proposed by Gorthi et al. [38]. In this study UML Use Case Activity Diagram is used to analyze impacts of changes and select test cases.

Minimization-Based Techniques

Minimization means the identification and elimination of redundant or obsolete elements of a group. Minimization for regression testing means elimination of irrelevant requirements that are not affected by code modifications as much as possible while providing maximum coverage of affected requirements. Minimization based regression test-selection techniques are applicable to a subset of previously verified test cases determined by using a modification-based technique. These techniques rely on finding a subset of test cases that provides the same coverage based on the impact analysis criteria. Qu et al. [39] developed an algorithm that minimize the test suite by maximizing concern requirement coverage and minimizing irrelevant requirement coverage.

Prioritization-Based Techniques

The prioritization based regression test-selection techniques are structured based on the sorting of test cases in increasing order according to determined coverage or cost values. Similar to minimization-based techniques, these techniques are applicable to a subset of test cases that are selected using a modification-based technique. These techniques are used especially when testers cannot afford to re-execute all of the selected test cases [40]. Testers select the test from prioritized list based on budget. Regression test cases can be prioritized based on factors such as fault history [41, 42] or customer priority. These factors can also be used as elimination factors for minimization of regression test suite. For instance, a value-driven approach named as PORT (Prioritization of Requirements for Test) has been proposed by Srikanth et al. [43] to prioritize test cases. In this study

prioritization of test cases is done based upon four different objectives: implementation complexity, requirements volatility, fault proneness of the requirements and customer priority.

2.2.2 Single Objective Selection vs. Multi-Objective Selection

Naturally regression test-selection process is an optimization problem and it aims to find best test suite option with regard to some criteria from some set of available alternatives. In real word regression testing scenarios, there are many different criteria that may have impact on effectiveness of a test suite like execution time, coverage, user priority, fault history etc. The use of a single criterion is not enough for regression selection and it may severely limits the fault detection ability of regression testing [44]. Despite this fact, no study was presented that take more than two objectives which mean criteria into account until the study of Yoo and Harman [10] in 2007. All previous studies focus on a single objective or two objectives that can be formulated as a single objective by using a ratio like coverage/execution time.

In the first study of that area, Yoo and Harman [10] use pareto efficient multi-objective regression test-selection approach with two and three different objectives which are execution cost, code coverage and fault detection history. They use three different algorithms to solve the two and three objective regression test selection problem: a re-formulation of the single objective greedy algorithm and two different version of genetic algorithm. In another study Harman [13] emphasize the importance of using multiple criteria in regression selection and provides a list that includes possible regression test-selection criteria. We use that study as the baseline to determine objectives of regression test-selection for the project analyzed in this thesis. Yoo et al [45] adopt the multi-objective search-based test suite selection technique proposed by Yoo and Harman [10] and apply it to an industrial project. Epitropakis et al. [44] performed an extensive empirical study to demonstrate the effectiveness of multi-objective approach for test case prioritization. In that study, three different objectives are considered to prioritize test case: average percentage of code coverage, average percentage of coverage of changed code, and average percentage of past fault coverage.

Because of the limitations of single objective regression test optimization in practice, there is an increasing preference of multi-objective approach in recent studies. But almost all of the multi-objective empirical studies stick in three objectives barrier. And although Harman [13] provides an extensive list of possible cost, value and constraint criteria, almost all of the related studies need source code access. The main aim of this thesis is reducing the cost of regression test suites with using of all possible criteria other than code coverage.

2.2.3 Multi-objective Regression Test Optimization (MORTO) Approach

In real world regression testing scenarios, testers face many different objectives and some of them are conflicting objectives. Therefore, industrial applicability of proposed single objective regression test-selection techniques in literature is limited [20]. Practical limitations of single objective approach prevent to handle real scenarios. Harman presents a manifesto for this problem in his study of Multi-objective Regression Test Optimization [13]. It proposes to handle regression test-selection process as a multi-objective optimization problem and different than other multi-objective based studies [10, 14, 45, 46] it provides a baseline for regression test-selection objectives. In that study, a list of possible objectives is given in two different groups: Cost based objectives and value based objectives.

Cost-based objectives

A cost objective can be defined as any item for which costs are being separately measured for each individual. The goal of regression selection process is to minimize these objectives.

Execution Time

Execution time of test cases is a basic natural criterion for regression testing optimization. Especially for short build cycles, it is clearly a restrictive concern for a tester. For this objective, test cases which have short execution times are privileged. For example, automated test cases usually consume less execution time compared to manual testing. Regression test-selection process should give priority to test cases with lower execution times.

Test Case Co-maintenance Effort

This objective means the effort required to update test case after a system modification. Since some of the test scenarios would be inapplicable due to changes. Regression test-selection process should give priority to test cases with lower co-maintenance effort.

Data Access Costs

System may need to interact with third parties to obtain data or services. For example, when testing service-oriented systems [47], accessing the systems of a third party may have a price. Because of this additional cost, tester may wish to eliminate test cases that need third party services. Regression test-selection process should give priority to test cases with the lowest data access costs.

Technical Resources Costs

Systems may require specific test environment for execution of some test case such as specific-purpose labs, controlled environment. Installation and preparation of this specific test environment requires designated resources that have a non-trivial cost. In order to reduce the cost of regression test suite, consumption of such resources should be reduced, while testing the system as fully as possible. Regression test-selection process should give priority to test cases with the lowest technical resources costs.

Setup Costs

Setup cost means the time and effort required for configuration of the certain devices, services, files or other aspects of the system in a particular way before the test execution. However, the set up costs for the test case may be significant in time and other costs. Such set up costs may also introduce dependencies, leading to an interaction between objectives and constraints. Regression test-selection process should give priority to test cases with the lowest setup costs.

Simulation (stub) costs

Some sub-systems or modules are expensive to be included directly in testing. Because of that a simulation or simulator for those objects are usually put in place. In these scenarios, implementing a simulation of system behavior may cause a significant cost. If the cost of

simulation can be assessed, it can be used an objective for regression selection. Regression test-selection process should give priority to test cases with the lowest simulation costs.

Value-based Objectives

These objectives denote test-related value (benefits) for each test case to be potentially selected for regression test suite. Generally, the goal here is to maximize these values as discussed below.

Code-based coverage

There are several code-based coverage features to be used in regression selection, such as branch, statement or mutation coverage. All of such goals reflect the objectives that want to be maximized, because they offer possible benefits in catching defects. Usually, the higher the coverage, the higher the defect detection rate is.

Non-code-based coverage

Non-code-based coverage means using system models or requirements as objectives in regression selection process. When compared to code-based coverage, non-code-based coverage objectives have been less used in literature. The reason of this is that models aren't used in software production systematically. If models are used in software production further, they can be used as much as code based coverage in regression test-selection.

Coverage of modified code

Covering modified code is an important objective for regression selection. In order to reduce number of test cases while covering as much modified code part as possible, a proper change analysis, control-flow and data-flow analysis should be conducted.

Coverage of impacted code/modules

In addition covering the modified code parts, regression test cases also need to cover the code parts or modules indirectly impacted by the changed code pieces. In order to use this objective in regression selection, a systematic change-impact analysis shall be conducted.

Reachability (activation), infection and propagation of suspicious code

Covering only faulty pieces of code may not be enough to reveal the defect. As it has been discussed in fault-based testing approaches, regression test cases should also stimulate the program state relating to the fault, and also execute the program lines after the fault such that the infected program state (as the result of the fault) propagated to program output or behavior.

Covering previously-suspicious or -faulty code

Using different fault localization techniques, failures found in previous execution of test suites can be used to determine a set of faulty pieces of code and a suspiciousness measure of different lines of code. It would be logical to prefer test cases which cover those faulty or suspicious lines of code in regression test suite.

Fault model sensitive

This objective means the sensitivity of regression testing approach to a fault model. If the subject system has a vulnerability or sensitivity to certain faults, this information can be used in regression test selection process so that those tests that reveal more faults in subject fault category can be primarily selected for regression testing.

Fault history sensitive

Different than the fault model, fault history uses the number of faults revealed by test cases. Prioritizing previously fault-revealing test cases that can be named as "proven performers" is a logical way of regression selection. There are fault history based studies in literature [42, 48, 49].

Human sensitive

There are important stakeholders of testing such as tester, developer, manager or customer. Those stakeholders may have their own opinions on prioritization of test cases based on their experience or feeling. Successful incorporation of test cases preferred by the stakeholders may be an important determinant of acceptance.

Business sensitive

Similar to the human factors, there are also distinctive and measurable objectives that have ability to test business features related to revenue generation or target-market acquisition. These objectives are wanted to use to prioritize test cases.

In addition to cost and value objectives, subject study provides a list of optimization constraints that can delimit the set of objectives. Regression selection process should take into account these constraints.

Precedence

This constraint means the need of execution of some test cases before others. Reason of this need may be a system state or a data level that is required for execution of dependent test case.

Conjunction Constraint

Counter to precedence constraint, some test cases may entail executing another.

Exclusivity Constraint

Different test cases may require the same resource and these tests cannot be performed simultaneously. For example; if a limited service provided by a third party organization is required for both test cases, regression selection process should select only one of them.

Dependence Constraint

Execution of a test case may reduce or increase the cost of another. For example, an excessive work which is required for a complex setup process of a certain test case may be reusable by another test case and may reduce the cost of test case.

Such constraints may be soft and they can be used as an objective or hard and they have to be applied directly to selection process. Besides Harman et al. states that "in most realistic regression testing scenarios there will be at least one cost- and one value- based objective, though there may be many others (and also constraints to respected)".

2.2.4 Applicable Approach and Techniques

The project is a large-scale project in acceptance phase in the aviation industry. Existing regression test-selection process totally depends on experience and preference of testers. Regression test scope is determined by mutual agreement of customer and manufacturer test teams.

There are two factors to determine the regression test selection methods for the project. One of them is source code access limitation. Because of this limitation, source code analysis and model analysis based regression test-selection techniques are not applicable to the project. Only requirement analysis based ones from modification analysis based techniques are applicable.

The second factor is the condition of full coverage of affected low level requirements. Prioritization techniques provide a list of test cases sorted based on determined objectives and some of the test cases from this sorting are executed based on time and cost limitations. This structure does not provide specific solutions that meet the full requirement coverage condition.

In order to meet the condition and limitation, a combination of requirement based selection and a minimization technique is determined for the project. Requirement analysis based selection phase guarantees the full coverage of affected requirements while minimization phase tries to find optimum candidate among the candidates emerged from requirement based selection phase based on available objectives.

In addition to these factors, there are multiple cost and value objectives affecting regression test selection process of the project. To use these objectives in regression selection, we focused on Multi-objective Regression Test Optimization (MORTO) approach and decided to apply this approach by using genetic algorithm. Genetic algorithm provides a basis for realization of requirement analysis based selection and multi-objective based minimization techniques.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

Because of the direct relation with practice, this study is industry-relevant research, in other words action research. In addition to producing research results and publish them as technical reports [15], action research needs interoperability between practitioners and researchers throughout the entire research process. This feature paves the way for application of ideas to real world problems.

For industry-relevant researches, a seven-step model seen in Figure 2-1 is defined by Gorschek et al [15] .

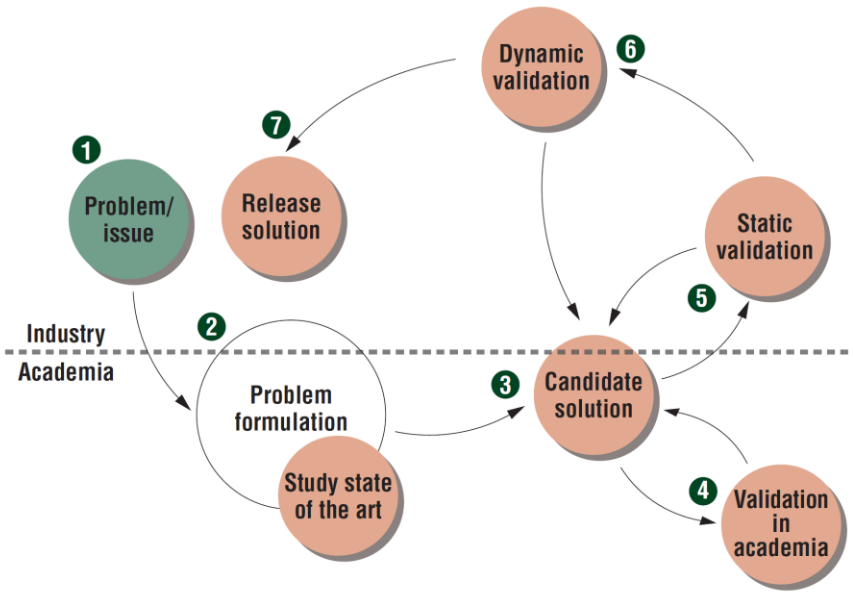

Figure 3-1 A model for industry relevant research[15]

Basic definitions of these steps are:

- Step 1: Determine possible improvement areas that mean evaluating existing practices, observing business and domain setting and finding out the demands imposed on industry.
- Step 2: Define a Research Agenda based on the prioritized needs determined in first step.
- Step 3: Formulate a Candidate Solution togather with industry.
- Step 4: Conduct Lab Validation means assessing results in controllable environments.
- Step 5: Perform Static Validation means presentation of solution to practitioners and get feedback from them
- Step 6: Perform Dynamic Validation means validation of solution in real environment but limited in scope.
- Step 7: Release the Solution means continuing with the actual implementation of model after evaluating the results from the static and dynamic validations.

In this study, we followed a tailored version of this model and attempted to realize these steps by using methodology defined in this chapter.

This chapter describes the research methodology of the thesis. It explains the research setting, research approach, research objectives and a suitable research methodology. Additionally, validity and reliability of the methods and the research limitations of the project are described.

3.2 Research Setting

This empirical study was applied to a large-scale project which has 2000 KLOC (Thousands of Line Of Code) in an unnamed company (details cannot be disclosed for confidentiality purposes). The project comprises of four Hardware Configuration Items (HWCIs) and eight Computer Software Configuration Items (CSCIs). HWCIs include 17 major hardware components. CSCIs include 63 major Computer Software Components (CSCs). The functionality of the system is defined by 935 system requirements and these requirements are verified by 54 different test cases.

The project is in acceptance phase and acceptance tests are conducted with participation of contractor and customer test teams. Customer side of subject project is a government defence organization. A test group is assigned to attend acceptance tests and this group is authorized to determine the regression coverage of each software build. On the other hand, contractor is a large defence company and its expertise spans in a broad range of products in defence. It has a specific test and evaluation directorate that support lots of different projects and their expertise covers a wide range of areas, including flight and ground testing of products, as well as laboratory testing. For all four areas of CMMI level assessment, the company has achieved Level 5 ratings: software and systems engineering, integrated product and process development, and supplier sourcing. As a result of having CMMI Level 5 rating, all test phases are conducted in a high level standard.

For the subject project, there is an independent testing group in the company. Additionally, another inter-projects group specialized in specific tests support subject project. The test group itself consists of several test teams, partitioned by the type of subsystems under test. All of the test activities conducted with participation of customer test team are black-box testing, since customer does not have source code access. Regression test selection process is conducted manually by evaluating affected requirement list and cost of the test cases based on the experience of test teams.

3.3 Research Approach

In this study, quantitative approach is used in which quantitative data is generated and processed. All required data to answer the research queations are collected and generated in quantitative form or transformed to quantitative form and processed in this form.

3.4 Research Objectives

The main aim of acceptance tests is successfully completion of all test cases. However, many test faults are detected during testing. After detecting a considerable amount of test failures, a new software build is submitted. In order to make customer side sure that software changes do not affect previously verified functionalities; contractor test team provides a list of possibly affected low level requirements and proposes a regression test suite based on their analysis and experience for each software build. After the evaluation of list and proposed test suite, customer test team who has no source code access and who does not have enough detailed information about software may accept the proposed test suite or may request adding some other test cases to it. Generally this process causes long discussions about size of the regression test suite. While contractor test team tries to minimize regression test suite because of the time and cost limitations, customer test team have tendency to maximize the number of the test cases to be on the safe side.

Both test teams are keen to improve regression test-selection practices to establish a systematic and cost- efficient regression test-selection process by using multiple cost and value objectives. In order to meet request of both test teams, solution method should cover all of the affected requirements and should provide multi-objective test selection.

Based on this analysis and motivations, the objectives of this study are to:

- Identify and describe the factors that affect regression test selection
- Determine an applicable and suitable solution and
- Specify the improvements provided by solution

3.5 Research Design

The research design is the overall plan or methodology of researcher to obtain answers to the research questions. Research methodology of this study is visualized in following diagram.

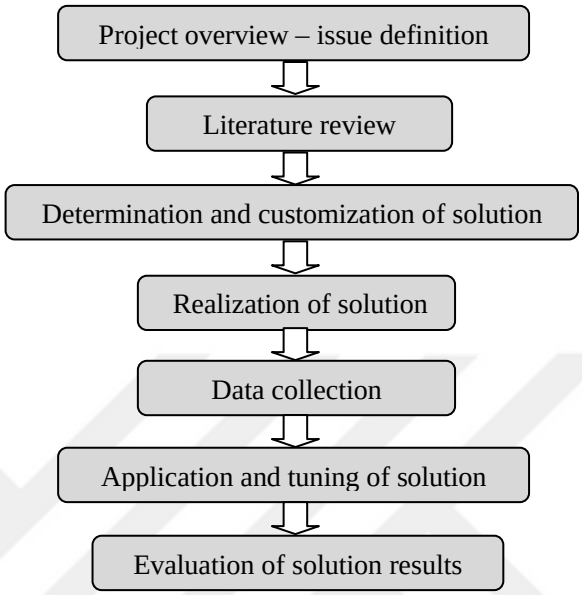

Figure 3-2 Research Methodology

Project overview - issue definition: In this step of methodology, existing processes in the project are reviewed and issue and challenges related with regression test selection process are defined. Limitations, constraints and objectives that will be part of solutions are determined with technical assesments and stakeholder input. Fundamentally, it is decided to find out a cost-efficient solution that meets the limitations, constraints and objectives of the project.

Literature review: Based on the issue definition, regression test selection approaches and techniques proposed in literature are reviewed. Advantages and disadvantages of them are analyzed and applicable ones are specified.

Determination and customization of solution: In this step, suitable and applicable methods and techniques that meets the requirements of issue are determined with participation of practitions. A solution approach is determined and a general solution is organized by using these methods and techniques. Then, determined solution is specificly customized based on the limitation and contraints of the project.

Realization of solution: This step includes the determination of algorithm structure and implemenation environment that will be used for realization of solution and implementation of solution. In order to make these decisions, applicable algorithms and implemenation environments are analyzed and applicable algorithm and implementation environment are selected. Then solution is implemented by using this structure.

Data Collection and application of solution: Required data is collected in two different ways. One of them is reviewing existing documents that were created in scope of the project. These documents are test procedures, test record sheets, test reports, system specification definition documents and fault database. Most of data is optained by using these documents. Second way of data collection is decision making meeting with participation of stakeholders. Objectives, weight of objectives, transforming of nominal level objectives data to rate level data which can be used in

calculations and some of the objectives' data are determined in these meetings by using Delphi method.

Tuning of solution: In order to reach optimum results in optimum time frame, determined parameters of imlemented algorithm are specifically tuned by using project data. This tuning process includes two different parts. In first part, each parameter analyzed seperately by using a value range proposed in literature. In second part, combinations of best parameter values are analyzed and best combination is determined after a number of executions.

Evaluation of solution results:In line with the aim of the study, cost efficiency of the solution is evaluated by comparing with the existing regression test selection method and another regression test selection method proposed in literature. Comparison is done in different cost aspects of regression test selection.

This methodology can be applied to similar projects by using below flow chart. This flow chart gives the project specific processes of the proposed solution.

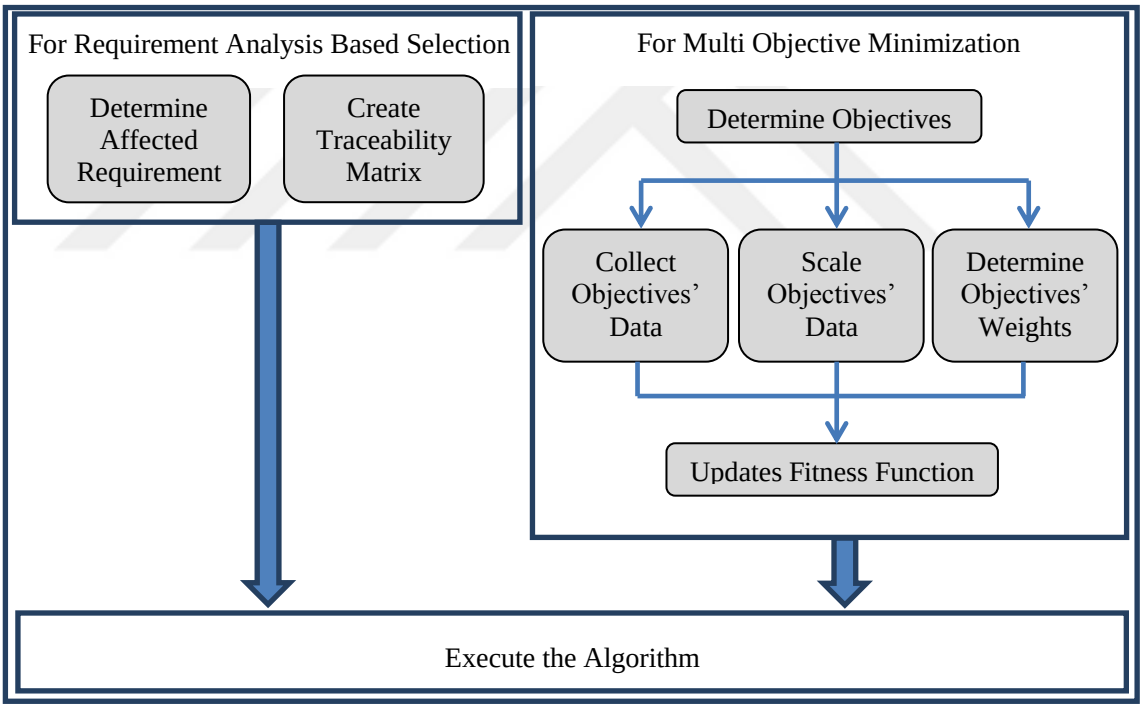

Figure 3-3 Application of Methodology

3.6 Validity Threats

The validity of a research indicates the reliability of the results in accuracy and objectiveness aspects. [66]. In scope of this research, except some part of data collection whole process is conducted by using quantitative approach in which quantitative data is generated and processed. On the other hand, some of required data is determined in decision making meeting by stakeholders. Subjectiveness of this data is the most important threat for validity of this research. In order to mitigate this threat, data collection process conducted in decision making meetings is done by the field specialists who have at leats five year experience in project. Additionally, this process is conducted by using Delphi method. The Delphi method is a reliable way to collect data with

consensus of a group of experts. In this method, experts answer the questionnaires in two or more rounds. This structure mitigates the threats of subjectiveness of data.

CHAPTER 4

DEVELOPMENT OF AN OPTIMIZATION ALGORITHM FOR MORTO

MORTO provides a flexible regression test-selection structure for different kind of software projects. Based on different combinations of cost, value and dependency objectives, MORTO can be operated as a source code analysis technique or a high level technique such as requirement analysis based technique or combination of both.

4.1 Choice of the Optimization Methodology: Genetic Algorithms

MORTO approach fundamentally proposes resolving regression test-selection process as a multi-objective optimization. Multi-objective optimization means optimization more than one objective simultaneously [50]. Classic linear algorithms solve multi-objective optimization problems by trying every possible solution and this requires quite much effort. NP-hardness in this computation makes heuristic search algorithms the only viable option for most of the complex optimization problems [51]. Solution produced by a heuristic algorithm may not be the best of all the actual solutions but heuristic algorithm would produce this solution in a reasonable timeframe.

There are several heuristic search algorithms defined in literature. The most common ones in these algorithms are genetic algorithm, simulated annealing, tabu search and ant colony [52]. Because of its scalability and flexibility [18], and because it has been widely used in the Search-based Software Engineering (SBSE)[53], we use genetic algorithm to empirically evaluate multi objective regression test optimization approach in this study.

4.2 Design of the genetic algorithm

This section describes the components of genetic algorithm that are specifically tailored to solve multi objective optimization algorithm for regression test-selection. We define genetic representation of solution structure in Section 4.2.1. Constraints determining legal chromosomes are explained in Section 4.2.2. Initial population creation of GA is discussed in Section 4.2.3. The fitness (objective) function is defined in Section 4.2.4. GA variation operators (crossover and mutation) are detailed in Section 4.2.5. And finally implementation of optimization algorithm is explained in section 4.2.6.

4.2.1 Representation of GA Chromosomes

As defined in Chapter 2, a number of chromosomes which equal to population size are created in each generation of genetic algorithm. Each chromosome represents a solution for subject problem. And as being in nature, chromosome is a set of genes that represents each independent variable in genetic algorithm. For multi objective regression test optimization, each chromosome represents a set of test cases which means a test suite and each gene in chromosome represents a test case.

In our study, binary representation is used and each independent variable which correspond a test case is represented as a bit which is the classical type of genes. The value 1 means existence of test case in solution, 0 means the opposite. Each solution which correspond a regression test suite is handled as an array of size M vector where M is the total number of test cases. The length of chromosomes which means the number of genes in the chromosomes is fixed and it is equal to the number of test cases that can take place in a solution.

4.2.2 Constraints

For this study, chromosomes that do not provide full coverage of requirements affected from software changes are not acceptable solutions. Each chromosome generated in genetic algorithm steps should satisfy full coverage of affected requirements. Therefore, we have to ensure that all genetic algorithm steps that produce solutions, initial population creation, crossover and mutation, always satisfy this constraint. In order to do so, full coverage of affected requirements is formally expressed in the context of our GA.

Implementation of this constraint using a traceability matrix is explained in section 4.2.6.

4.2.3 Initial Population

Population size means the number of individuals existing in a population. It is one of the most important parameter of genetic algorithm and determining the population size of a genetic algorithm is challenging. If the number of individuals is less than required, genetic algorithm can perform limited number of crossover and as a result of this only a small part of search space is explored. On the other hand, genetic algorithm may have performance problem if there are too many individuals. The optimal calibration of the population size for a given problem means that there should be a cost-benefit balance between convergence performance and accuracy [54]. In our genetic algorithm, we calibrated the population size using empirical guidelines in Search-based Software Engineering (SBSE) [18, 52, 53, 55]. We discuss in section 5.3 how we actually did the calibration for the case-study project.

Additionally, our initial population generation process ensures that the constraint defined in section 4.2.2 is met. This constraint negatively affects the performance of genetic algorithm, especially initial population creation process. In order to minimize that effect, initial population creation process is modified. A randomly selected test case that has requirement coverage more than average is added to each solution candidate created by initial population generation process.

4.2.4 Objective (Fitness) Function

In each cycle of genetic algorithm, a given number of worst solutions (chromosomes) are removed from population, as specified by crossover ratio, and same number new solutions are added to population to create a new generation. Selection of solutions to be removed is done by applying fitness function to each solution. Objective function, in other words fitness function, is the target function that needs to be optimized. It creates a fitness value which is the evaluation criterion for each solution.

Designing of fitness function is a critical process. If fitness function is not designed carefully, it may have performance or convergence problems (immature convergence or convergence difficulty). In addition to that, designing a fitness function for a multi-objective optimization problem is not as straightforward as it is for a single-objective optimization problem. There are several solution techniques for multi-objective optimization problem. As discussed in Chapter 2, they can be categorized in two groups: scalarization approach that transform multi-objective optimization problems to a single-objective optimization based on the preference determined before optimization process and Pareto approach in which a group of Pareto optimal solutions emerge and the most preferred Pareto optimal solution is selected based on the preference determined after optimization process. Pareto optimality means a balance point in resource allocation aspect in which it is impossible to improve resources usage of one individual without reducing at least one other individual [10].

Both approaches have advantages and disadvantages. Pareto approach may produce better solution than scalarization approach but it needs exponential size computation power based on the number of objectives. On the other hand, scalarization approach is the most preferred approach for Multi-objective approach because of its simplicity but it may not be successful as much as Pareto Approach in finding optimal solution. However, if preferences determined good enough, scalarization approach can reach one of the pareto optimal solutions.

In this study scalarization approach, weighted sum is used to solve the multi-objective regression test optimization problem because there are ten different objectives and pareto approach requires too much computational power than scalarization approach. Fitness function defined below is designed based on scalarization approach and cost and value objectives are transformed to one objective by using the factors that means the ratio of each objective in fitness value. These factors are determined by stakeholders of the projects. The heuristic underlying this is to find the optimum regression test set that minimizes fitness value.

$$\forall c \in Chromosome: OF(c) = \sum_{\forall g \in \text{Genes}} \sum_{\forall o \in \text{Objectives}} Value \times factor$$

Equation 1 - Fitness Function

In this formulation, OF*(c)* represents fitness function. Input of this function is chromosome and output is real number. In this formula, $Value$ means the collected data for each objective (value and cost objectives) of each test case (gene) and $factor$ means the predetermined ratio of related objective in fitness value. The $factor$ of each objective is defined by stakeholders of the regression

test-selection problem according to their preference. Defined objective function tries to find minimum fitness value. Therefore, $factor$ of value type objectives is negative real number while $factor$ of cost type objectives is positive real number.

4.2.5 Variation (Crossover and Mutation) Operators

Crossover and mutation are variation operators of genetic algorithm. Crossover means copying genes from parent chromosomes and creating a child chromosome. There are different crossover methods in literature. Based on results represents by the study [56], multi-point crossover which can be implemented as n-point crossover is used in this study and it is realized as 3-point crossover. Parents of each generation are inputs of crossover function and two children are created from each couple of parents based on randomly selected 3 crossover point as shown in Figure 4.1.

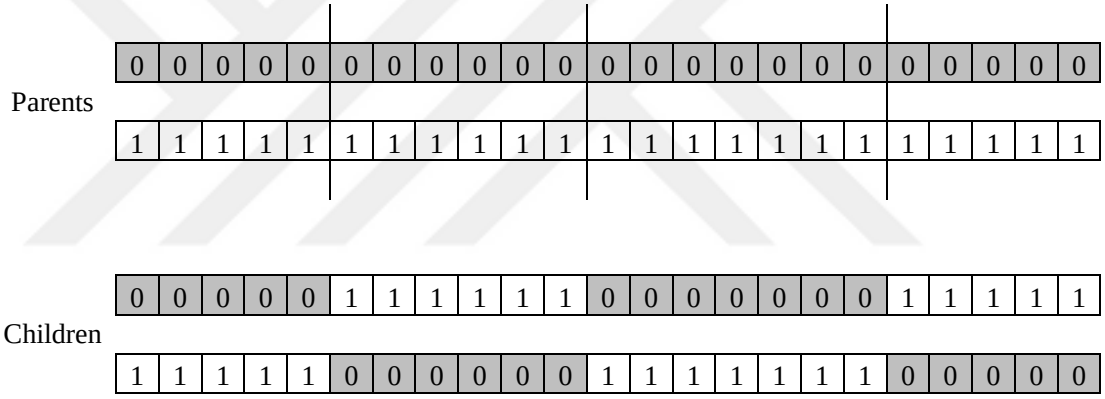

Figure 4-1 3-point Crossover

Mutation is another variation operator in genetic algorithm that is used to maintain genetic diversity from one generation of chromosomes to the next. There are different mutation implementation methods. Because of the structure of subject problem, bit string mutation is used in this study. In this mutation method, a number of bits determined according to mutation rate in random positions in a chromosome string are flipped as shown in Figure 4.2.

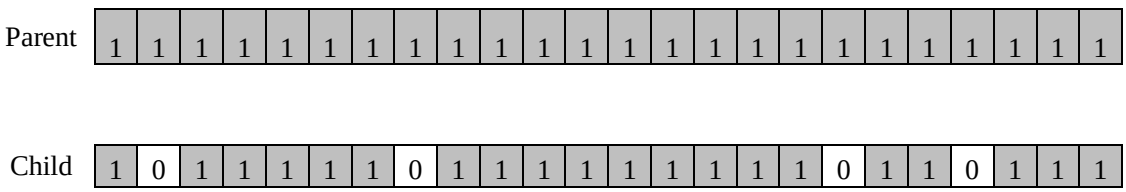

Figure 4-2 Mutation

In our genetic algorithm, crossover and mutation rates are empirically calibrated by using empirical guidelines in Search-based Software Engineering (SBSE)[18, 52, 53, 55]. Details of case specific tuning explained in section 5.3.

4.3 Implementation of the Genetic Algorithm

Optimization algorithm was implemented by using the genetic algorithm feature of the Matlab Global Optimization Toolbox. This tool provides a basic genetic algorithm structure with a variety of options. Additionally, it provides opportunity to customize each step of genetic algorithm such as replacing existing functions with new ones and by defining new criteria.

In this study, optimization algorithm is structured as two parts: Requirement coverage based selection and multi objective minimization.

Requirement Coverage Based Selection

Requirement Coverage Based Selection part basically guarantees that every solution candidate covers all of the affected requirements. This part is implemented as a constraint function and applied to initial population creation, crossover and mutation processes which are the solution creators of genetic algorithm as a validation statement. So, the constraint is applied to each solution created by the genetic algorithm and solutions that do not meet the constraint are discarded. This constraint increases the execution time of genetic algorithm especially initial population process, because initial population creation process creates candidates solutions randomly and probability of coverage of all affected requirements by randomly created solution is quite low.

Pseudo code of constraint is shown in Listing 4-1.

```
function output = Constraint ( candidateTestSuite, traceabilityMatrix, affectedRequirements
)
        coveredRequirements = candidateTestSuite * traceabilityMatrix
        if coveredRequirements contains affectedRequirements
            set output to valid
        Otherwise
            set output to invalid
end function
```

Listing 4-1 Pseudo code of requirement coverage constraint

Candidate test suite created by genetic algorithm, traceability matrix which provides requirement coverage of candidate test suites and affected requirement list created by developers are inputs of this constraint function. The traceability matrix contains which test case covers which high and low level requirements. By using traceability matrix requirement coverage of candidate test suite is found and if candidate test suite covers all of the affected requirements, output of function is set as valid, otherwise as invalid.

Multi-Objective Optimization Based Test-suite Minimization

This part of the algorithm is finding the fittest regression test suite among emerged candidates from requirement coverage based selection part. For this part;

- Fitness function defined in section 4.2.4 is implemented and
- Existing crossover and mutation functions are modified based on the selected mutation and crossover methods.

Fitness Function

In fitness function design defined in section 4.2.4, each objective value except the dependency is multiplied with a determined case specific objective factor. The results are collected in order to calculate fitness value of each gene which corresponds to a test case. Fitness value of a solution is calculated by collecting fitness values of test cases located in the solution and their dependent test cases.

In implementation phase, this process is realized as matrix products. Let M represent the number of test cases and N represent the number of objectives. First of all, an MxN size test case-objective matrix is multiplied with Nx1 size factors vector which includes the factor of each objective and an Mx1 size vector which includes fitness values of all test cases is obtained.

This calculation is done only one time and result vector is used throughout the execution. Then 1xM size candidate solution vector is multiplied with this Mx1 size fitness values vector and fitness value of the solution is obtained. It is the output of fitness function. Pseudo code of fitness function is shown in Listing 4-2.

```
function output = Fitness ( chromosome, testCaseObjectiveMatrix, objectiveFactors )
        fitnessValuesVector = testCaseObjectiveMatrix * objectiveFactors (one time)
        chromosomeFitnessValue = chromosome * fitnessValuesVector
        set output to chromosomeFitnessValue
end function
```

Listing 4-2 Pseudo code of fitness function

Crossover Function

As explained in section 4.2.5, the n-point crossover method is selected for this genetic algorithm design and it is realized as 3-point crossover. In implementation, 1xM size parent solution vectors (chromosome) are obtained from existing genetic algorithm structure. In each cycle of crossover function two parent solutions are used as inputs and two child solutions are created as output. In order to realize 3-point crossover, 3 different points smaller than M are determined randomly and parent vectors are divided in 4 parts from these points. First child solution is obtained by combining the 1^{st} and 3^{rd} parts of the first parent with the 2^{nd} and 4^{th} parts of the second parent. Second child solution is obtained by combining the remaining parts of the parents.

Pseudo code of crossover function is shown in Listing 4-3.

```
function output = Crossover( parentChromosomes )
        for each chromosome couple
                crossoverPoints =randomly selected 3 integer numbers smaller than length
of the  chromosome)
                child1 =  parent1(1-crossoverPoint1)&
                          parent2(crossoverPoint1-crossoverPoint2)&
                          parent1(crossoverPoint2- crossoverPoint3)&
                          parent2(crossoverPoint3- end)
                child2 =  parent2(1-crossoverPoint1)&
                          parent1(crossoverPoint1-crossoverPoint2)&
                          parent2(crossoverPoint2- crossoverPoint3)&
                          parent1(crossoverPoint3- end)
        set output childChromosomes
end function
```

Listing 4-3 Pseudo code of crossover function

Mutation Function

As stated in section 4.2.5, bit string mutation method is selected for this genetic algorithm design. In implementation, 1xM size solution vectors (chromosome) which will be subjected to mutation are obtained from existing genetic algorithm. These vectors and mutation rate are used as inputs of mutation function. In each cycle of mutation function one solution is used as inputs and one mutated solution is created as output. If the output does not provide full coverage of affected requirements, output is discarded and process is repeated until to reach an acceptable solution. For mutation, a number of bits determined according to mutation rate in random positions in a solution string are flipped.

Pseudo code of mutation function is shown in Listing 4-4.

```
function output = Mutation(Chromosomes, mutationRate )
        numberOfMutation = lengthOfChromosome * mutationRate
        for each chromosome
                mutationPoints =randomly selected numberOfMutation numbers smaller
than length of the chromosome
                for each mutationPoint
                        chromosome(mutationPoint)                                    =
bitReverse(chromosome(mutationPoint)
        set output mutateChromosomes
end function
```

Listing 4-4 Pseudo code of mutation function

Crossover and mutation rates are problem specific rates and traditionally determining what optimal rates of crossover and mutation are implemented by trial-and-error method [57]. Therefore, determining crossover and mutation rates is explained in parameter tuning section in Chapter 5. Other parameters used in genetic algorithm are given in Appendix-1.

The remaining parts of the genetic algorithm are directly used from existing genetic algorithm structure in Global Optimization Toolbox of Matlab.

CHAPTER 5

EMPIRICAL EVALUATION OF THE APPROACH

This chapter presents the application of genetic algorithm based MORTO explained in Chapter 4 in the industrial project. In this application, regression test suites are determined based on objectives determined by stakeholders among test suite candidates that provide full coverage of affected requirements. Furthermore, genetic algorithm parameters are tuned specifically to improve the performance of algorithm. Details of application are explained in following sections.

5.1 Goal, Research Questions and Metrics

Using the goal template of the GQM approach [58], the main goal of this empirical study is to investigate the current manual regression test-selection process of a large-scale industrial software project and improve regression test-selection by applying the above genetic algorithm-based MORTO approach.

Based on the above goal, following Research Questions (RQs) are raised:

- RQ1: How can the GA be best calibrated to yield the best results?

- RQ2: How much cost improvement does the multi-objective approach provide compared to the previous manual test selection approach and selective requirement coverage method proposed in literature?

These questions focus on providing improvements on existing manual regression test-selection method. RQ1 focus on fine tuning of genetic algorithm for a limited number of genetic algorithm parameters. RQ2 intends to compare two methods and prove the efficiency of multi-objective Regression test-selection method in cost perspective. In addition to manual regression test-selection method, MORTO approach is compared with selective requirement coverage method.

Metrics used to answer the RQs are listed in Table 5-1.

RQ	Metrics
RQ1	▪ Number of tuned GA parameters ▪ Execution time and fitness value ▪ Repeatability ▪ Convergence efficiency
RQ2	▪ Number of regression test cases (test suite size) ▪ Affected requirement coverage = Ratio of the number of affected requirements covered by a test suite to total number of affected requirements ▪ Number of irrelevant requirements (those which have not a known relation with software modifications) ▪ Fitness value

Table 5-1 Metrics used in the empirical study

5.2 Application of MORTO

In this section, application details of optimization algorithm defined in Chapter 4 are explained step by step and in order to show the efficiency of MORTO, a comparison between MORTO, existing regression test-selection process of the project and regression test selection method based on selective requirement coverage [14] is given.

5.2.1 Data Collection

In order to be able to apply optimization algorithm, required and possible data is collected. Different data collection methods are used for this empirical study. Some part of the data is collected directly from existing documents which are not prepared for the purpose of this empirical study or obtained from processing of data located in these documents. Other part is obtained from test team meetings which are conducted for the purpose of this empirical study. Since our approach has two components (requirement coverage based selection and multi-objective minimization), we discuss next the data collection for each of these components.

Requirement Coverage Based Selection

Requirement coverage-based regression test-selection part of solution uses the system requirements and their corresponding test cases, which are accessible by developer and testers [33]. Similar to the practices in many other companies, the company under study uses traceability matrices for associating requirements with test cases. We needed to collect that information to be able to trace the set of test cases which need to be rerun when a set of requirements undergo modifications.

Our requirement coverage based approach used the traceability matrix to derive connections between test cases, high level requirements and additional low level design requirements. It conducts selection process based on connections between these items. In this setting, high level requirements mean system level requirements representing functionalities of system. On the other hand low level requirements mean design level requirements representing the design decisions related with functionality. Due to confidentiality, unfortunately we cannot provide examples of the system requirement or design entities. These design level requirements are created from high level requirements.

For this part, traceability matrix and affected requirement list for each software versions are collected. Both of these data entities are obtained from documents created in the scope of the subject project. Additionally to make a comparison between existing regression test-selection method and MORTO, regression test suites determined with existing process for each software version are collected. It is also qualitative data and obtained from project documents.

Impact Analysis

Software Change Impact Analysis is defined as "The determination of potential effects to a subject system resulting from a proposed software change" [29]. In scope of regression selection, it can be used to find test cases that are affected by a set of changes.

Bohner and Arnold [59] classified impact analysis in two different groups: traceability and dependency impact analysis. Traceability impact analysis is a high level analysis and relies on links between requirements, specifications, design elements, and tests. Dependency impact analysis is more detailed (fine-grained) analysis and relies on relationships between parts, variables, logic, modules etc. In the absence of access to source code, dependency impact analysis cannot be conducted.

In this empirical study, Traceability Impact Analysis method is used. For each software version, software changes list and affected low level requirements list are created. Affected low level requirement list is the minimum bound for requirement coverage of regression test suite and it is used for validation of found solutions.

Traceability Matrix

Traceability Matrix is the fundamental structure of requirement coverage based selection part of application. It makes the complex relationship between test cases and the two different level requirement groups understandable and usable in regression test-selection process. The relationship between test cases and high level requirement list is many-to-many. On the other hand, the relationship between high level requirements and low level requirements is one-to-many. Traceability matrix provides completeness of the relationship between layers of information to determine the low-level requirement coverage of a test case.

Relational database structure for test cases and high level requirements has already been created in the scope of the subject project by the development team. In order to add low level requirements to traceability matrix, design documents and specifications were analyzed with the help of contractor and customer test teams. Traceability matrix includes 54 test cases and 935 low level requirements. An instance of traceability matrix is shown in Table 5-2.

Test Case	High Level Requirements	Low Level Requirements
3100x:gs-13-40	Gf-0070c1t3-x	142PX:1
4000x:gs-13-40	Gf-0074c1t3-x	181PX:0
4000x:gs-13-40	Gf-0074c1t3-x	185PX:0
4000x:gs-13-40	Gf-0074c1t3-x	184PX:0
4500x:gs-13-40	Gf-0083c1t3-x	125PX:4
4500x:gs-13-40	Gf-0084c1t3-x	194PX:0
4500x:gs-13-40	Gf-0084c1t3-x	201PX:0

Table 5-2 An instance of traceability matrix

Test cases, high level and low level requirements are symbolized with combination of numbers, letters and special characters. In order to use that matrix in genetic algorithm, it is transformed to a 54 x 935 (54 refers test case number, 935 refers low level requirement number) verification matrix. There is not any semantic difference between traceability matrix and verification matrix. This transformation just makes possible to use arithmetic operations on traceability matrix data. An instance of verification matrix is shown in Table 5-3. In this matrix, rows represent test cases and columns represent low level requirements (LLR). Matrix provides either the list of low level requirements tested in a specific test case or the list of test cases testing a specific low level requirement. The value 1 means that requirement stated in corresponding column is tested in test case stated in corresponding row, 0 means the opposite.

Test Case /LLR	125PX:4	142PX:1	181PX:0	184PX:0	185PX:0	194PX:0	201PX:0
3100x:gs-13-40	0	1	0	0	0	0	0
4000x:gs-13-40	0	0	1	1	1	0	0
4500x:gs-13-40	1	0	0	0	0	1	1

Table 5-3 Verification matrix

Affected Low Level Requirements

Affected low level requirement lists for each version of the System Under Test (SUT) are collected to use for control of requirements coverage of solutions created by genetic algorithm in population creation, crossover and mutation steps. Similar to process done for Traceability Matrix, affected requirement lists were transformed to 935 x 1 vectors to be able to use in genetic algorithm environment. The value 1 means existence of corresponding requirement in the changed requirement set in a given software version, 0 means the opposite.

Regression Test Suites Determined by Existing Regression Selection Process

In order to evaluate application of MORTO, regression test suites that were selected by existing regression test selection process used in the project are collected for each software version. They are also transformed to 54x1 vectors to be able to use in genetic algorithm environment. The value 1 means existence of corresponding test case in test suite, 0 means the opposite.

Multi-objective Optimization based Minimization

Minimization step is a multi-objective optimization of the possible regression test suites found in requirement coverage based selection step according to fitness value calculated with cost and value factors of test cases. For this part; ten different objectives are used for optimization. Objective data is collected for each test case. Most of objectives' data is obtained from documents which were previously prepared in scope of project, other part is newly created. Additionally, in order to use objectives' data in optimization calculations in genetic algorithm some transformation processes are applied to that data. After those processes objectives' data turn into a 54x10 matrix. Each row of this matrix represents a test case and each column represents an objective. Details of objectives' data and transformations applied to that data are explained in following section.

5.2.2 Selection of Objectives (Parameters) for the GA Fitness Function

As defined in Chapter 4, fitness function is used to evaluate each solution candidate in genetic algorithm and to determine worst and best solutions. Based on fitness value created by fitness function, n worst solutions (chromosomes) are removed in each generation of genetic algorithm and n new solutions are added to population to create a new generation. So, fitness function is the target function to be optimized.

Fitness function design is explained in Chapter 4. In order to apply this design to a specific case, two important decisions need to be made. The first one is determining objectives to be used as inputs in fitness function and the second one is the determining relation between these objectives to decide the factors of inputs and to compute outputs.

Objectives (Parameters of Fitness Function)

Most of the objectives used in this empirical study are determined or derived from the objectives defined in section 2.2.3. In order to find applicable objectives for minimization, regression test-selection objectives are evaluated with participation of stakeholders and a list of objectives is determined. Applicability means the availability of required data for subject objective or conformity of objective to subject system. Besides, one of the objectives is new and specific to this study.

Basically, there are two types of objectives: cost objectives and value objectives. In addition to these objectives, there is another objectives named as dependency. This objective corresponds to precedence constraint which is one of the optimization constraints defined by Harman [13]. Precedence constraint means that some test cases have to be performed before others to meet some conditions such as system state and data requirements. In our study, this objective includes test case labels which are prerequisite of another test case.

As defined in study of Harman [13]; based on the structure of specific regression selection problems, any combination of defined objectives and constraints can be formulated as a solution. In this thesis combination of below objectives is used.

For some of objectives (fault history), data has already been available in the project; but for others data is collected (execution time, setup cost) or created (test case execution priority).

Cost Objectives

These objectives corresponds the cost associated with each test case. In this study, five different cost objectives were applicable (and thus were chosen):

- Execution Time
- Third Party Cost
- Setup Cost
- Technical Resources Cost (Test Execution Environment)
- Oracle Cost and

Four of the cost objectives are used from the list defined in section 2.2.3. The fifth one, the oracle cost is a new objective.

Other than execution time objective, all cost data is just a label for each value located in each cost objective. This data is not usable for optimization calculations in genetic algorithm. In order to resolve this issue, cost objectives' data is transformed into unit of execution time which is staff-hour by using a stakeholders-created unification table which includes a factor for each different cost value located in each cost objective. That table is shown in Table 5-4. In decision-making meetings test team estimate the effort required for each cost type in staff-hour scale and factor for each cost type is obtained by using Delphi Method defined in section 2.1.6.

- ***Execution Time:***

Execution time is the time period required for test case execution. All test cases are manual tests and execution time data is collected from test record sheets that include the date, time, and participants' information. Some of the test cases were run more than one time within the scope of retest or regression tests determined by existing testing process. For these test cases, mean of execution time data is used. Unit of Execution time is staff-hour and it does not need a transformation to use in calculations.

- ***Third Party Cost:***

Third party cost means the service cost procured from third party organizations. In our subject system, some of the test cases require specific services that cannot be provided by merchant or customer. Third party needs for each test case are collected from prerequisites parts of test procedures. There are five different third party services in collected data and names of services are written in Table 5-4.

Some of third party services are procured by merchant, while others are procured by customer. This separation of procurement is not taken into account.

- ***Setup Cost:***

Setup cost means the effort required for readiness of system to execute the test cases. In subject system some of the test cases require devices, services, files or a specific mode of the system.

This data is collected from prerequisite parts of test procedures by test team. There are three different setup cost types in collected data and they are written in Table 5-4.

- ***Technical Resources Cost (Test Execution Environment):***

For our subject system there are different test environments (software on simulated hardware lab, software on real hardware lab, real system or any combination of these environments) and usage of these environments cause different level costs. Lab installation costs are not subject of this study because installation of labs is done only once.

There are six (software on simulated hardware lab corresponding SIL, software on real hardware lab corresponding HIL, real system or any combination of these environments) different technical resources cost types in collected data and they are written in Table 5-4.

- ***Oracle Cost:***

Verification of some requirements cannot be completed by successfully execution of related test case. They require analysis of data collected during test. This analysis process cause additional cost other than test execution which is named as oracle cost.

Oracle cost data is collected from verification part of test procedures. There is only one type of oracle cost and it is written in Table 5-4.

Value Objectives

Value means usefulness or worth of something. Value-based objectives mean that any item or subject that provides measurable benefits. In this study, four different value-based objectives are used for regression selection:

- ❖ Number of Detected Faults (Cumulative)
- ❖ Number of Detected Faults (Last version)
- ❖ Faults Priority and
- ❖ Test Case Execution Priority

The first three objectives are obtained from fault history of five different versions of project software that are created within the scope of acceptance test period. These objectives' data is ratio level data and it can be used in optimization calculations in genetic algorithm. The last value objective (Test case execution priority) is created by stakeholder based on the experience obtained from previous test sessions.

- ***The Number of Detected Faults (Cumulative)***

Experience has shown that if the software is developed and upgraded periodically, reemergence of similar faults is highly possible [60]. So fault history of a test case is a value and giving priority to test cases which have detected more faults in earlier stages can be a good practice. In other words, selecting test cases that show better fault detection performance in test history can increase the affectivity of test suite in fault detection context.

Cumulative number of detected faults for a test case is the number of faults detected throughout the acceptance period. It is obtained from fault database.

- ***The Number of Detected Faults (Last Version)***

In addition to the cumulative number of detected faults, number of faults detected in last version provides awareness about the recent fault detection performance of a test case. Because of this contribution, it is added as a separate value objective to objective list. It is also obtained from fault database.

- ***Fault Priority***

Fault priority, is obtained by analyzing of priority information of faults. There are three priority values defined in fault database: 1, 2 and 3. The value 1 among the priority levels is the most important one. For each test case, previously detected faults are grouped based on priority level. And by applying below formula which is defined in decision-making meetings with consensus of stakeholders, Fault Priority value (*FP*) of a test case is calculated. Number of faults in a priority level is shown as *#PriorityLevel*.

$$FP = (3*\#1) + (2*\#2) + (1*\#3)$$

- ***Test Case Execution Priority***

Test case execution priority value of a test case means its importance in test team's point of view based on their experiences.

Test case execution priority is created by test team by using Delphi Method for each test case based on their experience and their sight about the power of test case scenario. It is an ordinal level data and there are three priority levels: 1, 2 and 3. The value 1 is the most important one. In fitness value calculations, they are used respectively as 1, 0.66 and 0.33.

- ***Dependency***

Dependency for a test case means that another test case is a prerequisite for it. In order to execute a test case, its prerequisite one has to be executed before. So, prerequisite test case has to be added to proposed test suite, if it is not already in the test suite. This objective corresponds to precedence defined by Harman [13].

Dependency data is collected from prerequisite part of test procedures. It is nominal level data and it includes label of a test case. For optimization calculations in genetic algorithm, corresponding value in subject solution vector (chromosome) is transformed to 1. This process adds precedence test cases to solution.

The Identifications of Factors of Objectives (Ratios of Parameters) in Fitness Function

As defined in Chapter 4, in this study scalarization approach, weighted sum, is used for solving multi-objective regression test optimization problem. This approach is the simplest method for evaluating a number of alternatives but it is applicable only when all the data are represented in the same unit.

In order to provide unity of cost-based objectives, we use the converting table shown in Table 5.4. This table is created by consensus of test team. Unit of execution time which is staff-hour is used as baseline. Enumeration column represents subtypes of each cost type. It is nominal level data and each number in this column corresponds to a different subtype. For example Technical Resources Cost has six subtypes. Staff Hour Factor column presents the cost equivalence in terms of staff-hour. These factors for each cost type are created by using Delphi Method in decision-making meetings. Effort required for this convertion was not taken into account, because this process is done one time and it can be eliminated. This column provides the ratio level forms of nominal cost objectives' data.

	Enumeration Code	Real Value	Staff Hour Factor
Technical Resources Cost	1	Lab (Software)	4
	2	Lab(Software + Hardware)	8
	3	Lab(Software + Hardware)+Lab (Software)	12
	4	Real	24
	5	Lab(Software + Hardware)+Real	32
	6	Real+Real	48
Third Party Cost	1	Simulation Support	2
	2	Technical Support	8
	3	Equipment Service	12
	4	Flight Service	32
	5	Satellite Service	40
Setup Cost	1	Special Setup	2
	2	Special Disk Preparation	8
	3	Scenario Generation	40
Oracle Cost	1	Post Test Analysis	8

Table 5-4 Unification of cost-based objectives

On the other hand, units of value-based objectives are not suitable to convert staff hour. Three of them are inferred from fault history and other one which is test case execution priority (human sensitive) is created by stakeholders. In order to provide intended weight of all objectives in fitness value calculations, all objectives except test case execution priority are scaled between 0 and 1 by using below formula:

$$scaledValue = Value/MaxValue$$

Because units of all cost-based objectives are converted into same unit, *MaxValue* inferred from entire cost data is used for all cost-based objective scaling. For value-based objectives, MaxValue is inferred separately from each objective data. Besides, "Test Case Execution Priority" objective data is structured in opposite way. There are only 3 different level priorities as 1, 2 and 3 in ordinal level and the value 1 is the most important one. So, for that objective data is sequentially transformed into 1, 0.66 and 0.33.

After scaling entire data between 0 and 1, weighting table shown in Table 5-5 is used for calculating fitness value. That table provides the transformation of multi-objective regression test-selection problem into single objective structure. Weight values in this table are created in decision making meeting by using Delphi Method. As done in unification of cost objectives; effort required

for this weighting process was not taken into account, because this process is done one time and it can be eliminated. Weight values of value-based objectives are transformed into negative, because genetic algorithm structure provided by Matlab Global Optimization Toolbox is developed to find minimum fitness value. But, value-based objectives are benefit objectives and the higher the values are, the better it is. By multiplying weight values of value-based objectives with -1, this issue is resolved.

	Execution Time	Third Party Cost	Setup Cost	Oracle Cost	Technical Resources Cost	#Detected Fault (Cumm)	#Detected Fault (Last Version)	Fault Priority	Test Case Execution Priority
Weight	0.1	0.1	0.1	0.1	0.1	-0.15	-0.05	-0.2	-0.1

Table 5-5 Weighting table

Although dependency is an objective, different than to above nine objectives it represents a test case in fitness value calculation perspective. Because of that, it is handled in a different way. Its representing test case is added to solution by converting its value in solution vector to 1. Therefore number of weighted objectives is nine.

5.3 RQ1-Empirical Tuning of the Genetic Algorithm

5.3.1 Tuning of Genetic Algorithm

There are empirical studies [18, 52, 55] in literature show that proper tuning of a GA has a major impact on its performance and finding appropriate parameter values is one of the challenging aspects of tuning a GA. Main components of genetic algorithm are population size, variation operators (mutation and crossover) and the selection operators (parent selection and survivor selection) [61]. In this empirical study, to be able to reach optimum solution in optimum time frame the most important genetic algorithm parameters, population size and variation operators (crossover and mutation rates) are tuned with limited scale experimental comparisons. Genetic algorithm structure provided in Matlab Global Optimization Toolbox has many other parameters that can be tuned. However, because the combinations of parameters increase exponentially in parallel with the number of parameters, a limited number of parameters are used for GA calibration. For other parameters, we used the default values defined by Matlab Global Optimization Toolbox. The list of parameters and their values are written in Appendix-1. In addition to parameter tuning, initial population creation step of genetic algorithm is tuned.

Population Size

In literature, there are several studies that suggest adequate population size. For example, De Jong [62] proposed a population size ranging from 50 to 100 individuals. Grefenstette et al. [63] offered a range between 30 and 80 in his study, while Schaffer and his colleagues [64] suggest a smaller population size, between 20 and 30. In this study; algorithm was run 100 times for each population size value start from 5 to 100 with an increasing value 5 for three different cases. Each case means a different regression test-selection problem for different affected low level requirements list. In order to compare scanned area for each population size value, we used mean of execution time and mean of fitness value for 100 runs. Results as execution time and fitness value curves are shown in Figure 5-1, 5-2 and 5-3.

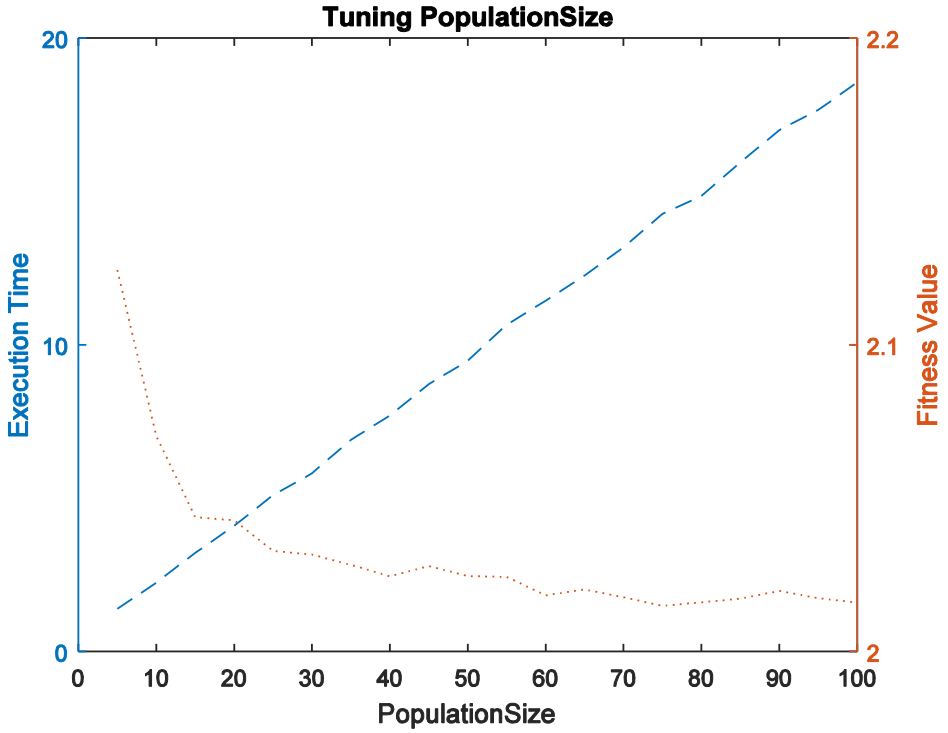

Figure 5-1 Population size effect on fitness value and execution time – requirement set 1

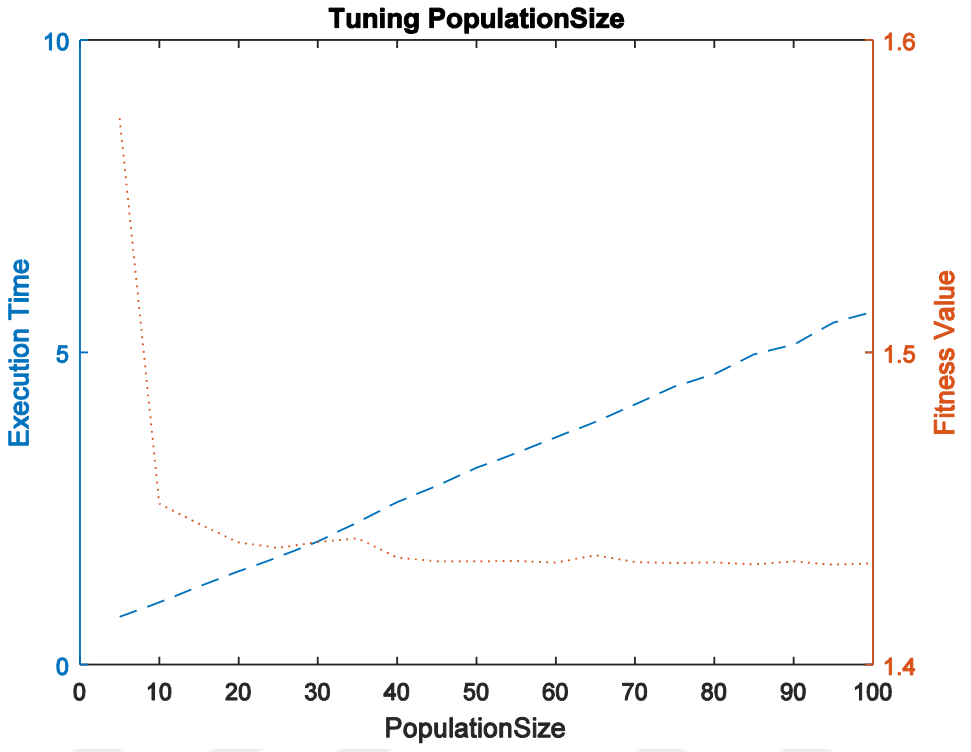

Figure 5-2 Population size effect on fitness value and execution time - requirement set 2

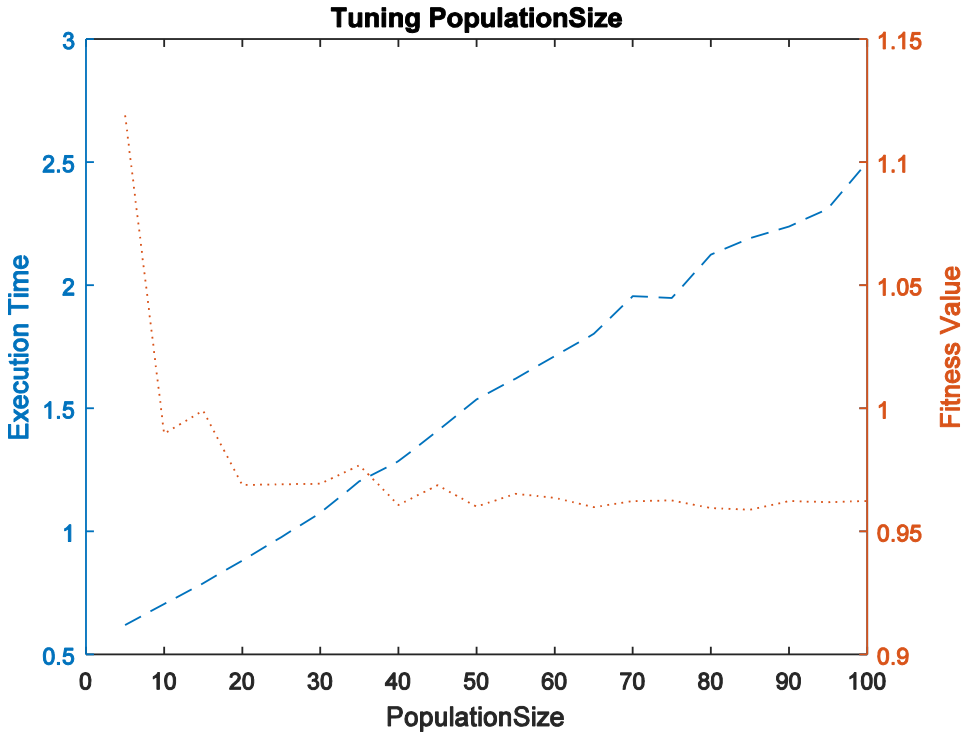

Figure 5-3 Population size effect on fitness value and execution time – requirement set 3

In each figure, there are two curves for each affected low level requirement sets. After analysis of these curves, it is figured out that fitness value converge a minimum plateau after population size 30 and in parallel with population size execution time increases linearly. Based on this analysis, 30, 40, 50, 60, 70 and 80 values are evaluated with combination of other parameter settings in below.

Variation Operators

Crossover Rate

There is a common usage of crossover rate between 0.5 and 1 in literature. Crossover rate below 0.5 may decrease the fitness quality [57]. As done in population size tuning, same setting is used for crossover rates between 0.5 and 1 with an increasing value 0.02. Results as fitness value and execution time curves are shown in Figure 5.4, 5.5 and 5.6.

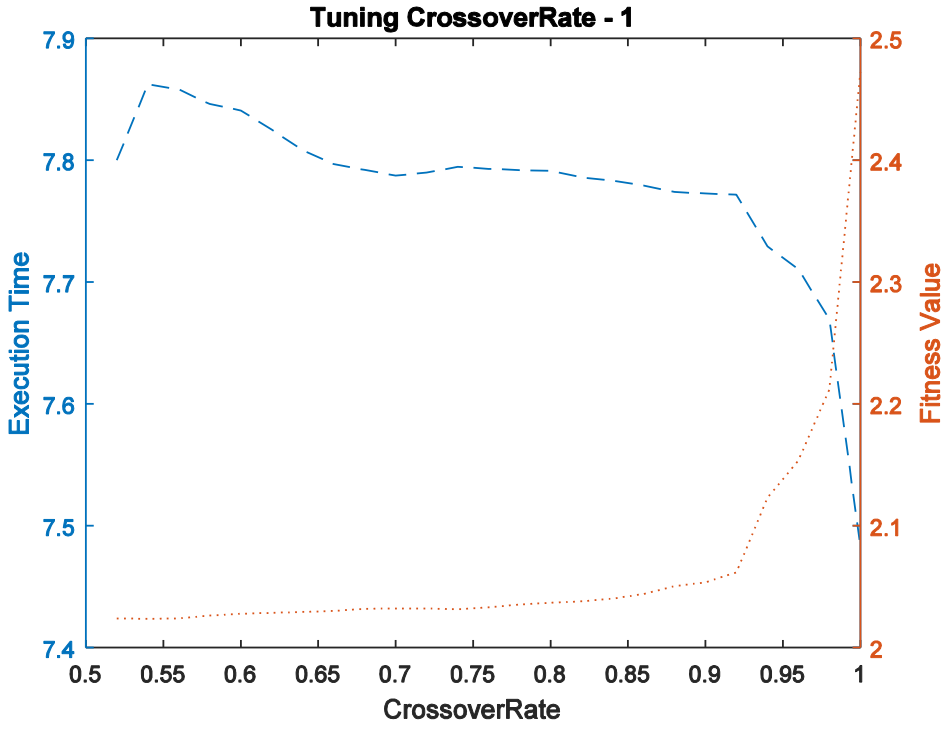

Figure 5-4 Crossover rate effect on fitness value and execution time – requirement set 1

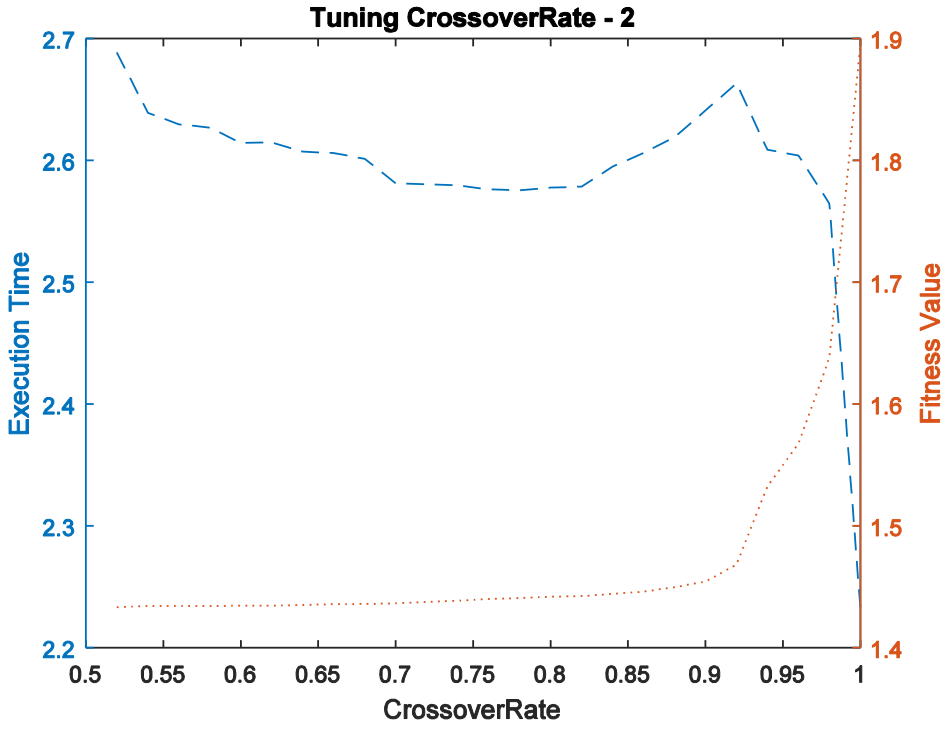

Figure 5-5 Crossover rate effect on fitness value and execution time – requirement set 2

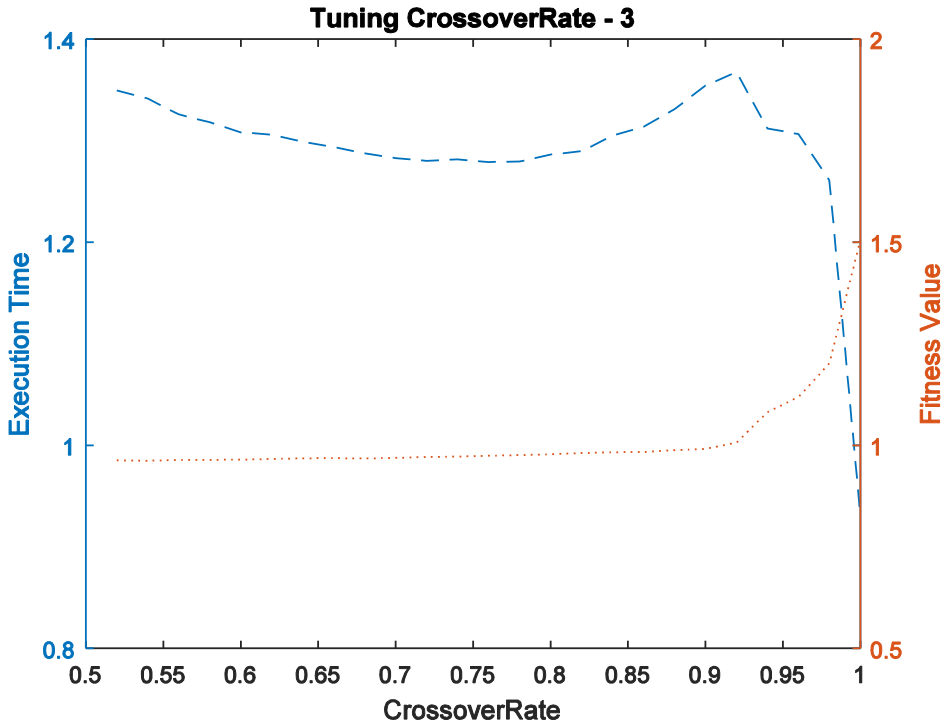

Figure 5-6 Crossover rate effect on fitness value and execution time – requirement set 3

In each figure, there are two curves for each affected low level requirement sets. After analysis of these curves, it is figured out that fitness value remains on a minimum plateau between 0.5 and 0.9 crossover rate values and also these values have no remarkable effect on execution time. Based on this analysis; 0.5, 0.6, 0.7, 0.8 and 0.9 values are evaluated with combination of other parameter settings in below.

Mutation Rate

On the other hand, mutation rate should be a low probability in respect to crossover rate. If mutation rate is too high, the search will perform as a primitive random search [57] and genetic algorithm may stall in converging. In literature, there is a tendency to propose quite small mutation rate. For instance, De Jong [62] suggests a mutation rate of 0.001, Grefenstette [63] suggests a rate of 0.01, while Schaffer et al. [64] formulated the expression 1.75 / λ length (where λ denotes the population size and length is the length of chromosomes) for the mutation rate. Mühlenbein [65] suggests a mutation rate defined by 1/length. Based on these proposals, each mutation rates between 1/54 and 27/54 (this means between 0.01 and 0.5) with an increasing value 1/54 are executed 100 times (54 is the length of chromosome). Results as fitness value and execution time curves are shown in Figure 5.7, 5.8 and 5.9.

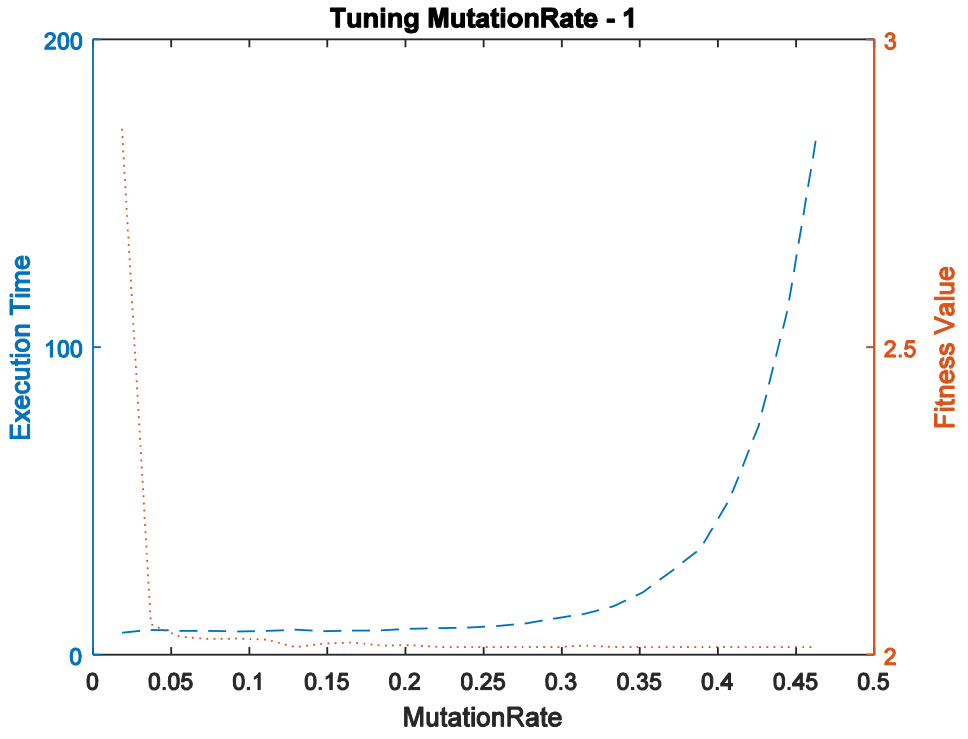

Figure 5-7 Mutation rate effect on fitness value and execution time – requirement set 1

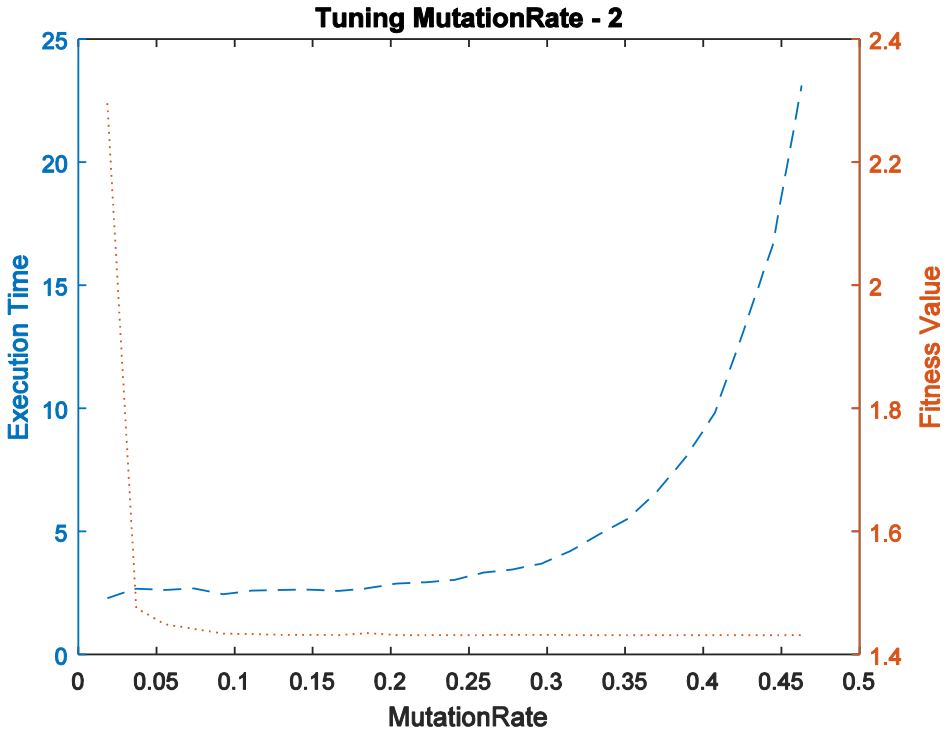

Figure 5-8 Mutation rate effect on fitness value and execution time – requirement set 2

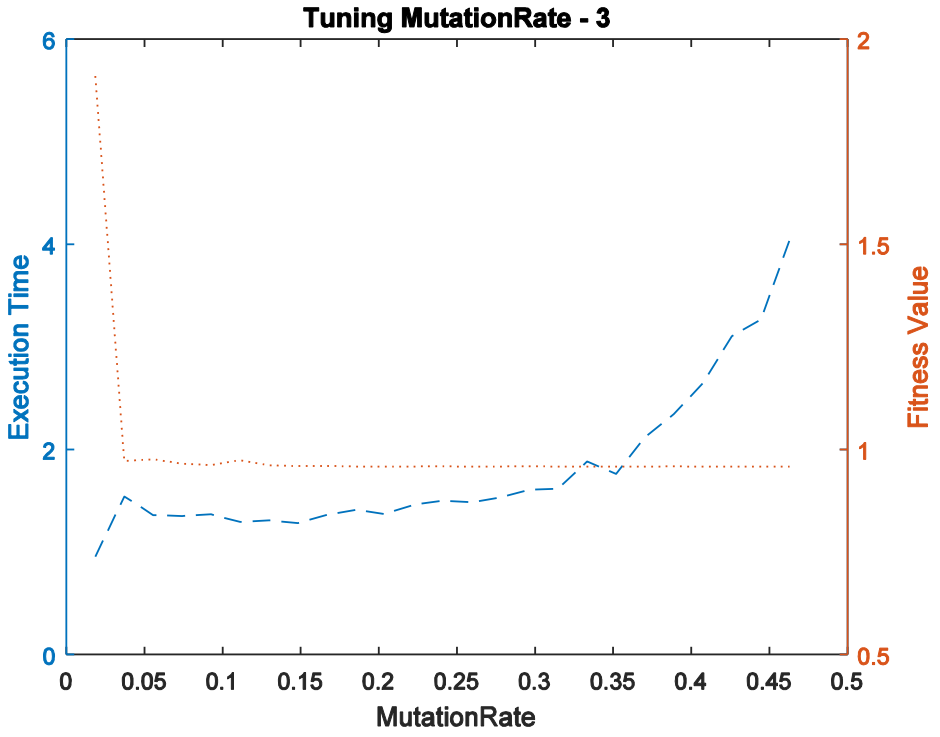

Figure 5-9 Mutation rate effect on fitness value and execution time – requirement set 3

In each figure, there are two curves for each affected low level requirement sets. After analysis of these curves, it is figured out that fitness value reaches a minimum plateau after mutation rate 4/54 (0.074) and keep on it throughout analysis space. On the other hand, in parallel with mutation rate execution time increases exponentially. Therefore; mutation rates 4/54, 5/54, 6/54, 7/54, 8/54 and 9/54 (between 0.074 and 0.166) values are evaluated with combination of other parameter setting for tuning.

In addition to these parameters, termination criterion which defines when to stop the search needs to be decided. As it is shown in Figure 2-2 Genetic Algorithm Flow Diagram [19], search process in genetic algorithm is repeated until a termination condition which is the limit of search budget has been reached. Matlab Global Optimization Toolbox provides a number of termination criteria:

- Total number of generations
- Time limit
- Fitness limit
- Total number of stall generations
- Stall time limit
- Stall test
- Function Tolerance
- Nonlinear constraint tolerance

Any of above termination criteria or any combination of them can be used as search budget for application of genetic algorithm. However, termination condition should avoid needless computations and prevent premature termination. In our empirical study, we used combination of stall generations with function tolerance and total number of generations. The algorithm stops if there is not a relative improvement in fitness value more than function tolerance over 20 stall generation or when total number of generations reaches 120. Total generation number limit is decided after analysis of a number of executions. In these executions, it is figured out that almost all executions end without reaching 100 generations and with stall generations limit, 120 generations provide an acceptable search space.

In order to tune genetic algorithm parameter, genetic algorithm is run 20 times for 3 different cases with each combination of below parameter values which are determined based on the individual analysis of the population size, crossover rate and mutation rate parameters. In total 6 x 5 x 6 x 20 x 3 = 10800 experiments are analyzed.

- Population size: 30, 40, 50, 60, 70, 80
- Crossover rate: 0.5, 0.6, 0.7, 0.8, 0.9
- Mutation rate: 4/54, 5/54, 6/54, 7/54, 8/54, 9/54

In these experiments, parameter settings are compared based on fitness value and three specific settings are focused: worst, best, default. The worst combination is the one that give the maximum fitness value out of the 10800 experiments depending on chosen termination criteria. Similarly, best represents the minimum fitness value. The "default" combination is set to 80 for population size, 0.8 for crossover rate and 1/54 for mutation rate. These values are in line with the values commonly proposed in the literature. This default setting values are chosen before execution any of the experiments. Worst, best and default fitness values for three different cases are shown in Table 5-6.

	Worst			**Best**			**Default**		
	Case1	Case2	Case3	Case1	Case2	Case3			
Population Size	30	30	30	40	60	40	80		
Crossover Rate	0.5	0.6	0.5	0.7	0.8	0.7	0.8		
Mutation Rate	6/54	4/54	5/54	8/54	7/54	7/54	1/54		
Fitness Value	2.12	1.48	1.09	2.01	1.43	0.95	2.21	1.64	1.37

Table 5-6 Tuning results

Parameter combination which provides best fitness value is determined as tuned parameter configuration for this empirical study. In this study, for the first case best parameter configuration provides 5% smaller fitness value than worst parameter combination and 9% smaller than default parameter combination, for the second case 13% smaller than default and 3% smaller than worst, for

the third case 30% smaller than default and 13% smaller than worst. In fitness value perspective, tuning process provides limited improvement between worst and best parameter combinations. The reason of this limitation is selecting best parameter values in first step of the tuning that includes individual analysis of parameters. On the other hand, parameter tuning provides a valuable improvement between default and best parameter combinations. Based on this analysis, it can be stated that defining a default parameter setting for all cases is not possible. Genetic algorithm parameters should be specifically tuned for each different case. As a result of this analysis; 40 as population size, 0.7 as crossover rate and 7/54 as mutation rate are determined for subject case study.

Initial Population Creation Tuning

Full coverage of affected requirement constraint that is applied to every solution increases the execution time of genetic algorithm especially initial population process. Because initial population creation process creates candidates solutions randomly and probability of coverage of all affected requirements by randomly created solution is quite low. In order to improve execution time performance of initial population creation step of genetic algorithm, a randomly selected test case which has a requirement coverage more than average is added to each solution if it not already in the solution. Performance enhancement gained with this modification is shown in Figure 5.10.

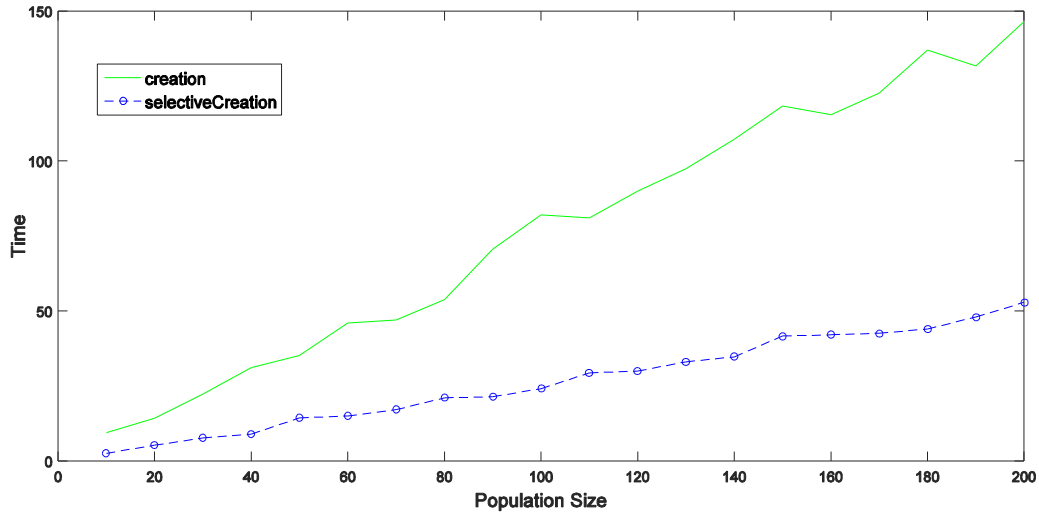

Figure 5-10 Creation vs ModifiedCreation

In this figure, "Time" shows the execution time of population creation process, "Population Size" shows the number of individuals (each individual means a possible solution) exists in population. Unit of time is second and this data is obtained from 100 runs of algorithm for each population size. Average of 100 runs for each population size is used as time value.

5.3.2 Assessing GA's Performance

As a heuristic search algorithm, genetic algorithm does not guarantee to find the solution that has minimum fitness value (global minimum) and its performance and outputs can vary across multiple runs. Therefore its results need to be validated. For this empirical study, results of genetic

algorithm are validated based on repeatability of GA results for multiple executions and convergence efficiency throughout the generations towards a minimum.

Repeatability of GA results for multiple executions:

Repeatability means the stability and reliability of genetic algorithm results. In order assess the repeatability of genetic algorithm used to realize MORTO, it is executed 250 times and variability of the average or best chromosome's fitness value is investigated. Descriptive statistics of the fitness values for three different cases are shown in Table 5-7.

Cases	Min	Max	Average	Median	Standard Deviation	Fitness Values	Percentage
1	2.01	2.28	2.02	2.01	0.03	2.01	%69.6
						2.02	%14.8
						2.06	%11.6
						2.07	%2.8
						2.28	%1.2
2	1.43	1.55	1.43	1.43	0.01	1.43	%77.2
						1.43	%19.6
						1.49	%2.8
						1.55	%0.4
3	0.95	1.24	0.96	0.95	0.04	0.95	%67.6
						0.96	%21.6
						0.97	%6.8
						0.98	%1.6
						1.23	%1.2
						1.23	%0.8
						1.24	%0.4

Table 5-7 Statistics of fitness values

Average and median values are very close, thus indicating that the distribution is almost symmetric. For the first case, 5 different values; for the second case, 4 different values; for the third case, 7 different values are generated and genetic algorithm reaches the minimum fitness value for most of the runs. This analysis shows that genetic algorithm results obtained in this empirical study are stable and reliable.

Convergence efficiency throughout generations towards a maximum:

In this empirical study, convergence efficiency is the second criterion to assess validation of genetic algorithm results. It is a useful feature to assess the selected mutation and cross-over operators, population size and chromosome representation. In this empirical study; in order to assess convergence efficiency, the number of generations required to reach a minimum fitness plateau is analyzed. Distributions of generation numbers required to convergence for three different cases are shown in Figure 5-11.

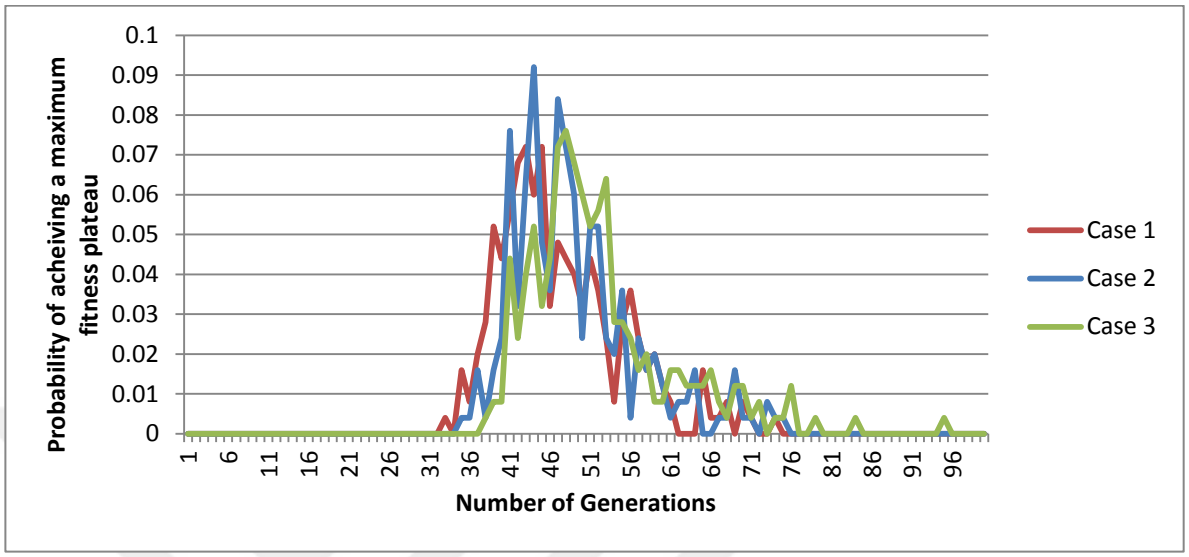

Figure 5-11 Generation numbers to reach a minimum plateau for three different cases

The statistics of these generation numbers over 250 runs for three different cases are shown in Table 5-8. The minimum and maximum values are 33 and 95 and for all three cases average generation number is 50. It can be stated that 50 generations are required to converge to the final result. The variation around this average is limited and in worst case genetic algorithm used in this empirical study can reach a minimum fitness plateau less than 100 generations. This number is in line with the experiments reported by Garousi [18].

Cases	Min	Max	Average	Median	Standard Deviation
1	33	74	48	45.5	7.7
2	35	75	49	47.5	7.7
3	38	95	52	50	8.9

Table 5-8 Statistics of generations required for convergence

5.4 RQ2-Improvement in regression test cost

This section of thesis responses the second research question of the study and in order to show the cost improvement in regression test-selection process a comparison between existing manual regression test-selection approach and genetic algorithm based MORTO is provided in test suite size, affected requirement coverage, irrelevant requirement coverage and fitness value perspective. Also MORTO is compared with another regression test selection method proposed in literature, selective requirement coverage based regression test selection [14], in test suite size, affected requirement coverage, irrelevant requirement coverage fitness value and execution time perspective. This method was chosen because the main constraint for regression test selection process of the project is source

code access limitation. This method is a high level method which does not need source code access and all data required for this technique is available to compare. It is realized by using genetic algorithm and 0.5 is used as weighting factor (α) explained in section 3.2.

In order to demonstrate efficiency of application model of MORTO; test suites selected by subject approach are compared with the test suites selected by manual regression test-selection process currently used in subject project and test suites selected by the selective coverage based regression test selection method proposed by Gu et al. [14]. Comparisons are done for five different version of project software configuration. Due to improbability of executing regression test suites found by application model of MORTO and selective requirement coverage method, fault detection based comparison cannot be done. The reasons of this improbability are the change on acceptance period of project and close down of the software-in-the-loop (SIL) and hardware-in-the-loop (HIL) laboratuaries as a result of management level decisions. However, scenarios of test cases that are proposed by MORTO are reviewed and it is manually confirmed that all test faults detected during execution of manually selected regression tests can be detected by MORTO-proposed test cases.

For all of the regression selection processes, covered requirements are separated into two groups as affected requirements and irrelevant requirements. The first group contains the affected test requirements which have a relation with modifications; the second group contains the irrelevant test requirements which have not a known relation with modifications. Relations between affected requirements and code modifications are defined by developers. Ideally all of the affected requirements should be covered by proposed regression test suites, while the irrelevant test requirements should be covered at minimal level. But it is hardly possible to cover all affected requirements by using manual test selection method, because it is an np-complete problem and it needs a non-deterministic algorithm for solution. Both selective requirement coverage based regression test selection method proposed by Gu et al. [14] and application model of MORTO structured in this thesis provide heuristic search-based solutions.

5.4.1 Evaluation of the resulting test suite

Each affected low level requirements list is used as a feasibility criterion (selection) in application model of MORTO and fittest test suite (minimization) according to fitness value for each build is found by genetic algorithm based on defined objectives. Because of the feasibility criterion, each test suite provides a 100% affected low level requirement coverage and irrelevant requirement coverage of these test suites are found by using traceability matrix. On the other hand, coverage of affected and irrelevant requirements is used as an objective in fitness value calculation of selective requirement coverage method.

In order to make a successful comparison between three different regression test selection methods, affected and irrelevant requirement coverage of test suites selected by existing regression test selection approach and selective requirement coverage based regression selection method are found also by using traceability matrix. And then test suites selected by existing regression test selection approach and selective requirement coverage method are used as inputs in fitness function of application model of MORTO and fitness value of each test suite is found.

5.4.2 Analysis of Results

Comparison is done based on the test suite size, affected and irrelevant requirement coverage, and fitness value of test suites selected all three regression test selection methods. Additionally execution times of selection process of MORTO and selective requirement coverage based regression test selection methods are compared. Although test suite size and irrelevant requirement coverage are not direct objectives of proposed solution methodology, they are used in comparison of manual, selective requirement coverage and MORTO methods to provide an overview. Details of comparison are explained in following sections.

Test Suite Size

Test suite size means the number of selected test cases in test suite. Figure 5.12 shows the comparison of three techniques based on test suite size. MORTO provides smaller test case number for three versions of project software than existing regression test selection approach. In one of other two versions, they have same test case number and in last one existing approach provides smaller test case number. On the other hand MORTO provides smaller test case number for two versions of project software than selective requirement coverage method. In one of other three versions, they have same test case number and in other two, selective requirement coverage method provides smaller test case number.

Because test suite is not a direct objective for MORTO and selective requirement coverage method, results may change for different executions. To make an evaluation as to which approach is better, test suite size alone is not enough. It needs to be supported with requirement coverage and fitness value of test suites.

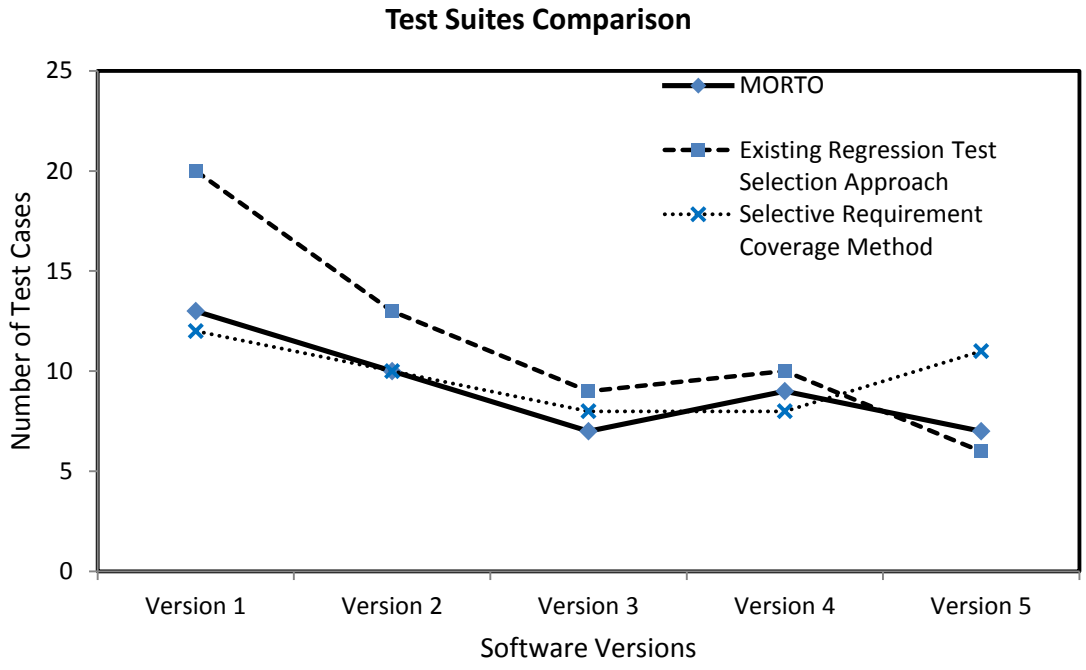

Figure 5-12 Test Suites Comparison

Requirement Coverage

In this empirical study requirement coverage of five different versions of subject software are compared based on affected requirement coverage and irrelevant requirement coverage of three methods. The results are shown in Figure 5-13 and 5-14. The graphics are provided for illustration and qualitative evaluation. For complete quantitative data, see Table 5-9. This table provides entire comparison data for all software builds.

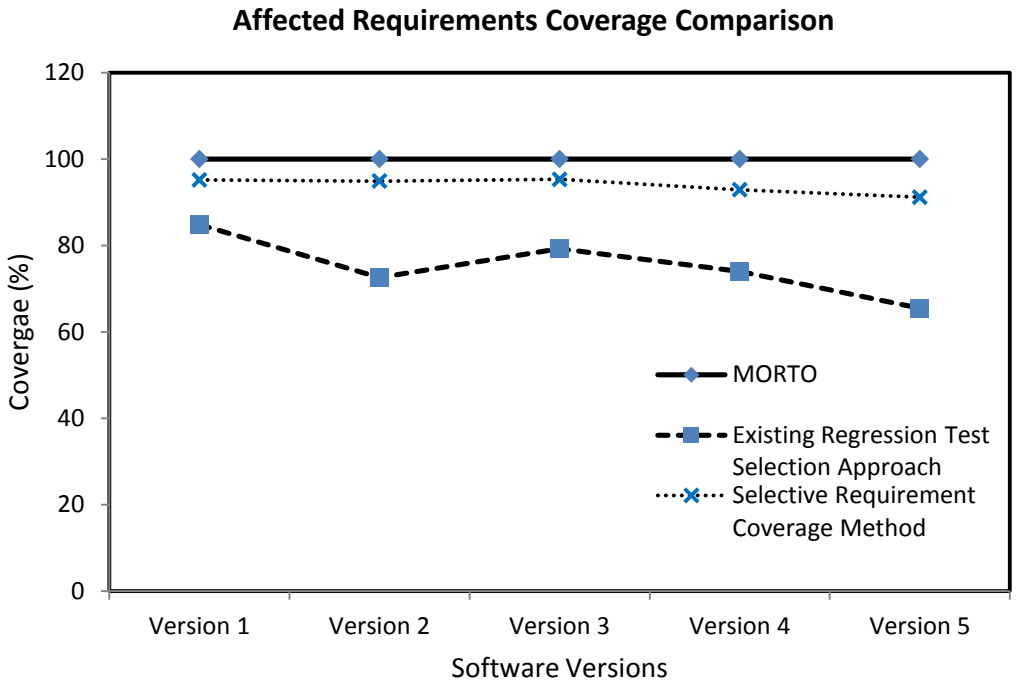

Figure 5-13 Affected requirements coverage comparison

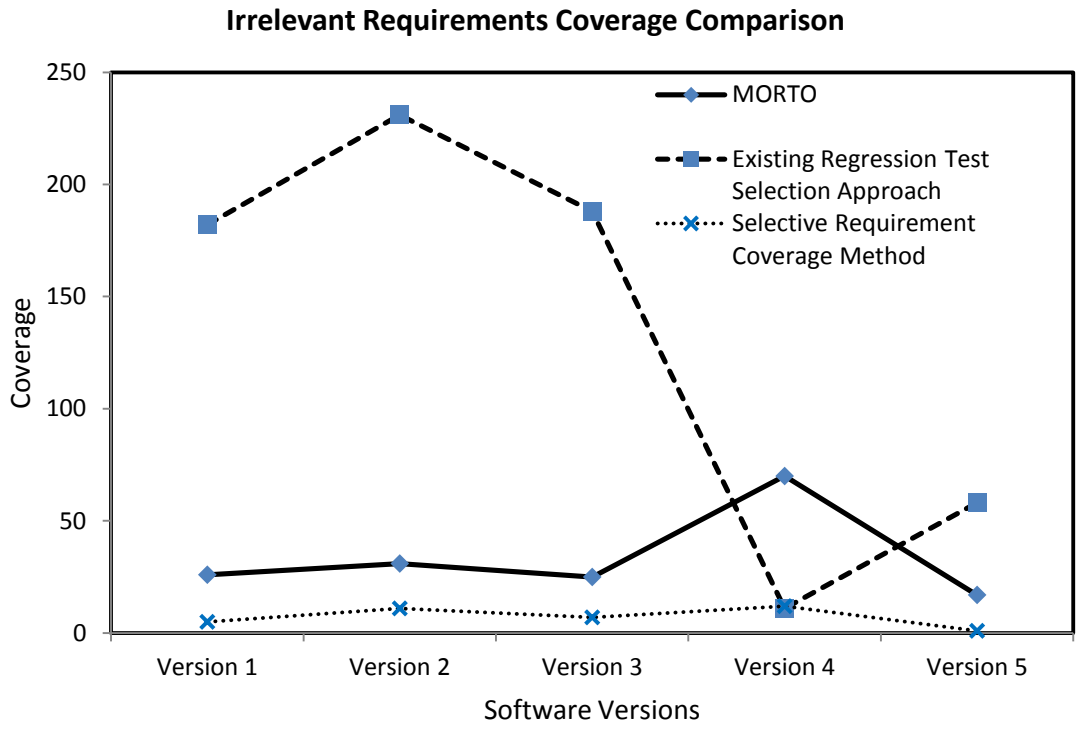

Figure 5-14 Irrelevant Requirements Coverage Comparison

Methods	Items for Comparison	Version 1	Version 2	Version 3	Version 4	Version 5
Existing Regression Test Selection Approach	# of Test Cases	20	13	9	10	6
	# of All Req.	445	459	356	136	132
	# of Affected Req. Covered	263	228	168	125	74
	# of Irrelevant Req. Covered	182	231	188	11	58
	Fitness Value (cost-value)	3.31	2.33	1.5	1.73	1.0
MORTO	# of Test Cases	13	10	7	9	7
	# of All Req.	336	345	237	239	130
	# of Affected Req. Covered	310	314	212	169	113
	# of Irrelevant Req. Covered	26	31	25	70	17
	Fitness Value (cost-value)	2.01	1.43	0.9	1.26	0.89
Selective Requirement Coverage Method	# of Test Cases	12	10	8	8	11
	# of All Req.	300	309	209	169	104
	# of Affected Req. Covered	295	298	202	157	103
	# of Irrelevant Req. Covered	5	11	7	12	1
	Fitness Value (cost-value)	3.07	1.94	1.3	2.07	1.5

Table 5-9 Methods comparison data

The empirical study results for requirement coverage show that compared with the existing regression test selection approach and selective requirement coverage method, MORTO provides better (full) affected requirement coverage. But because of the total coverage constraint, execution time of MORTO is quite long than selective requirement coverage method especially for affected requirement numbers bigger than 60 as shown in Figure 5-15.

In irrelevant requirement coverage perspective, selective requirement coverage method is the most successful one because irrelevant requirement coverage is used as a direct fitness value objective in this method. On the other hand, MORTO has less irrelevant requirement coverage than existing regression test selection approach for four of software versions. For only one software version, existing regression test-selection approach provides less coverage than MORTO. It is not also a surprise result because irrelevant requirement coverage is not a direct objective used in genetic algorithm. In this study, cost objectives which can be formulated as time and effort are used for optimization. Irrelevant requirement coverage is not

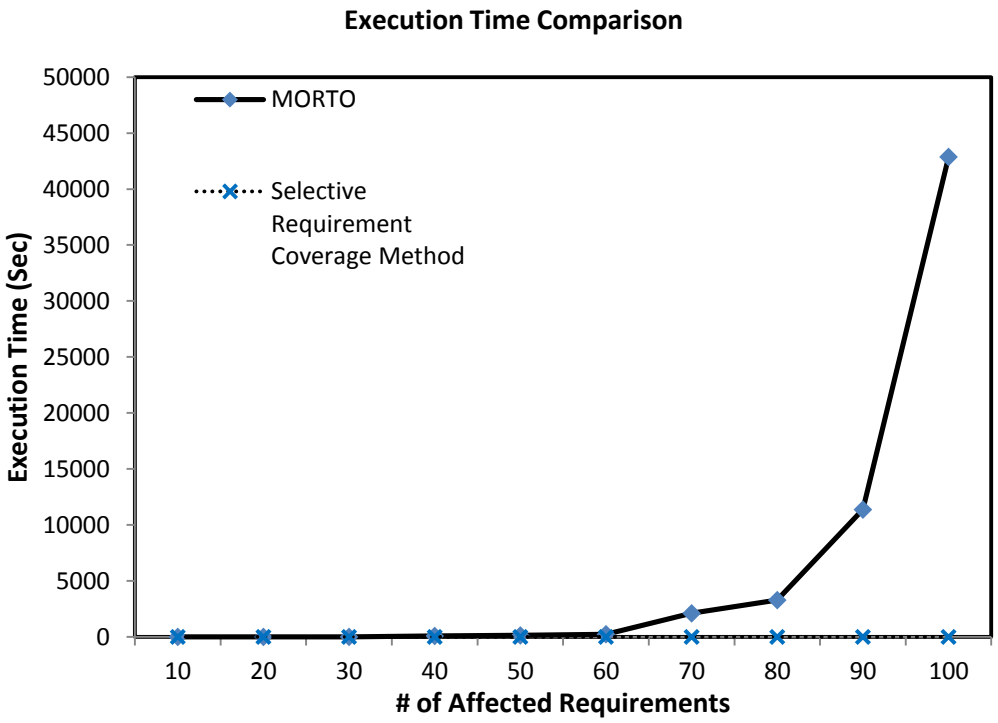

Figure 5-15 Execution Time Comparison

Fitness Value

Fitness function for application of MORTO is defined in section 5.2.2. Fitness values of each test suite of each software build for all three methods are calculated by this fitness function to compare. In this study, fitness value is the one and only determinant for solution selection. Because of that it is the most important factor to compare in scope of this study. It means the cost-benefit value of solutions. For example one of the methods may provide better solutions in scope of test suite size or irrelevant requirement coverage but if cost-benefit values of these solutions are worse than other methods', that method will not be a good choice in scope of this study.

Figure 5.16 shows the comparison of MORTO with other two methods based on fitness value. In these comparisons, it is seen that fitness values of MORTO are less than other methods. That means minimization part of MORTO provides more efficient test suites than existing regression test selection approach and selective requirement coverage method.

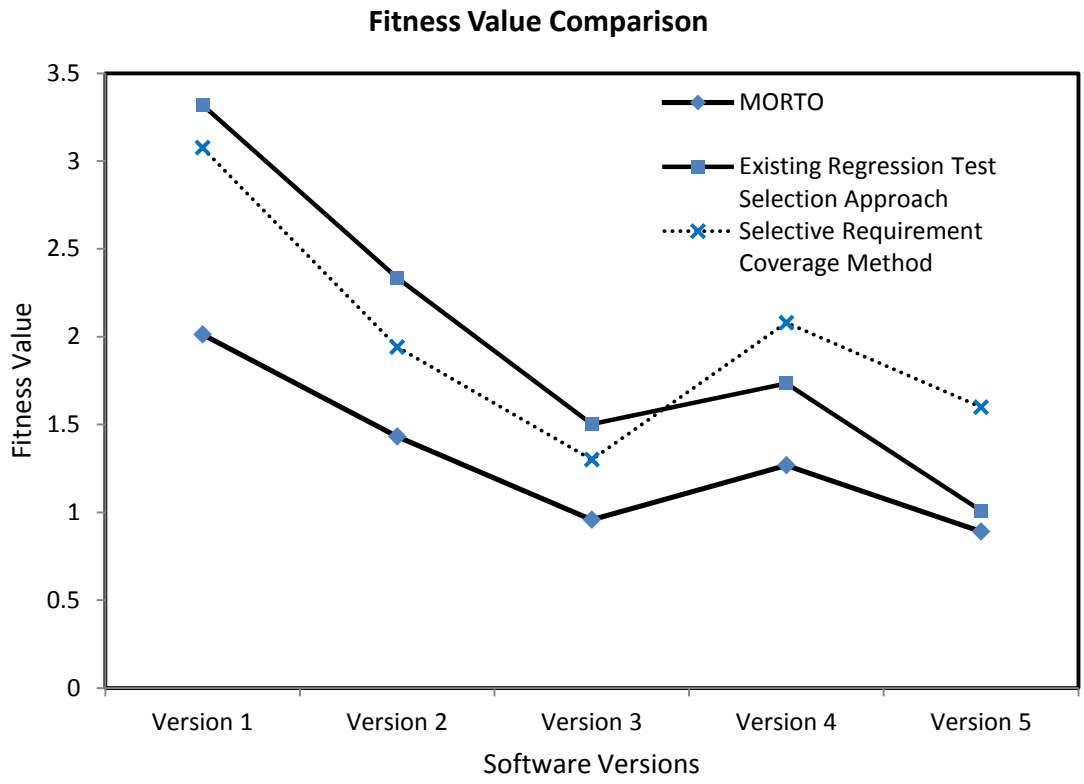

Figure 5-16 Fitness Value Comparison

5.5 Discussions of the results

The overall goal of this empirical study is to improve the existing manual regression test-selection process of an industrial project with source code access limitation. Most of the regression test-selection techniques proposed in literature are source code analysis based and they are not applicable for subject project. Furthermore, these techniques require deep and time consuming static analysis in source code to map changes to tests and they are not applicable to large-scale and complex systems. Therefore, instead of code based analysis; a faster, high-level analysis is better suited for this study.

From another point of view, most of the regression test-selection techniques presented in literature are structured based on a single objective. However, in real word regression testing scenarios, there are many different criteria that may have impact on effectiveness of a test suite like execution time, coverage, user priority, fault history etc. The use of a single criterion does not answer regression test selection requirements in practice and it severely limits the fault detection ability of regression testing.

5.5.1 Implications for the industrial partners

As a result of these assessments, a customized model of MORTO approach is used for regression test-selection in this thesis. This model comprises of two phases: requirement coverage based regression test-selection and Multi-objective regression test minimization. First phase reveals a set of test suite candidates that provides full coverage of affected requirements by using a traceability matrix. This matrix provides high level and fast analysis of links between requirements and test cases. Second phase of the methodology aims to find optimum set of test cases among solution candidates emerged in first step by using different objectives in genetic algorithm.

In order to demonstrate the effectiveness of application model of MORTO; test suite size, requirement coverage and fitness value results are compared with existing regression test selection approach and selective requirement coverage method for five different software versions in empirical study part of this thesis. From affected requirement coverage and fitness value perspective, MORTO is fairly successful. On the other hand, there are fluctuations in test suite size and irrelevant requirement coverage comparison between methods. Reason of these fluctuations is clear that test suite size and irrelevant requirement coverage are not fitness value objectives of multi-objective minimization phase of MORTO. They can be added as an objective to multi-objective minimization part of solution methodology based on the problem environment for future case studies.

Besides; in this empirical study fault detection is not used as an evidence of effectiveness for MORTO, because running selected test suite for each software version with corresponding software configuration is technically improbable for this study. The reasons of this are explained in section 5.4. This circumstance can be evaluated as a risk of diminishing fault detection rate. However, fault detection performance of test suites proposed by MORTO are manually reviewed based on the faults detected during execution of manually selected regression tests and it is found that for this specific case there is not diminishing risk of fault detection rate. Furthermore; for future studies that will use MORTO as a baseline, fault detection can be used as an evidence of effectiveness.

5.5.2 Implications for the research community

Optimization objectives and features used to demonstrate effectiveness of MORTO are determined based on project specific environment factors and available data. Because of this, drawing certain conclusions for general regression test-selection problems may not be meaningful. But application model of MORTO can be used for any kind of regression test-selection problem with some modifications based on the conditions of problem. For example; if source code access is possible, a source code analysis based modification technique can be used for selection part of model or minimization objectives can be customized based on available data.

5.6 Limitations and threats to Validity

This empirical study is an action research that uses search based optimization for solution method. As the purpose of the study is to apply a tailored version of a technique to an industrial project and show the effectiveness of proposed methodology by comparing with existing process and another regression test selection method proposed in literature. Design method of this study is ‘one factor with two treatments’. Factor is regression selection and the treatments are the proposed solution, other regression selection method and, the existing regression selection method (control

group). Dependent variables for this experimental study are affected requirement coverage and fitness value.

The validity of a study indicates the reliability of the results in accuracy and objectiveness aspects. [66]. In this section, four types of threats to validity are analyzed and discussed based on structure defined by Wohlin et al [66]. Additionally, steps taken to minimize or mitigate these threats are discussed.

5.6.1 Internal Validity

Internal validities mean specifically demonstration of differences made by an experimental treatment/condition and supporting that demonstration with sufficient evidences. In this perspective, this experimental study demonstrates the differences with a two group design in which one of the groups is exposed to a treatment while other one which is a control group is not exposed to the treatment. In order to realize the effects of treatment, both groups are tested similarly and results are compared.

Existence of a control group mitigates most of the internal validity threats (history, maturation, testing, statistical regression, selection, mortality). On the other hand, there are two possible threats to internal validity in this experiment: Poor parameter setting and lack of discussion on code instrumentation. Poor parameter setting threat is minimized by explicitly explaining selected parameters for proposed solution and for comparison of two regression selection processes. On the other hand, lack of discussion on code instrumentation is minimized by using a ready structure provided by a tool (GA structure provided by Global Optimization Toolbox). Modifications in that ready structure are explicitly explained.

5.6.2 Construct Validity

"Construct validities are concerned with issues that to what extent the object of study truly represents theory behind the study" [66]. In this empirical study, all but one threat to the construct validity are not considered harmful. Exception is lack of assessing the validity of effectiveness measures. For regression test-selection concept, fault detection rate is an important criterion and because of the technical improbabilities, fault detection rate could not be used as an effectiveness measure in this study. The reasons of this are explained in section 5.4.

In order to mitigate this threat, effectiveness measurement is extended as covering test suite size and irrelevant requirement coverage which do not take part as an objective in regression selection process.

5.6.3 Conclusion Validity

Conclusion validity threats are dealt with the relationship between treatment and outcome. Existence of statistical relationship between the parts that are involved needs to be figured out. [67] In conclusion validity perspective, possible threat is 'Random heterogeneity of subjects" for this empirical study. As a heuristic search algorithm, genetic algorithm, is used for solution structure, initial population of genetic algorithm is randomly generated. In case of a single run, this random generation may affect the results of heuristic search. In order to mitigate that threat, all experiment is

executed several times for each instance and most important genetic algorithm parameters (population size, crossover rate and mutation rate are tuned for this study.

5.6.4 External Validity

External validity is dealt with generalization of the results of a study. Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice. As mentioned before, this study is an action research. It presents a solution for a specific industrial problem with specific parameters or settings. However, solution provides a general and flexible structure which is applicable to most of the regression test selection problems by customizing the structure based on limits and constraints of the problems. In order to mitigate this threat, all case specific parameters and tuning processes proposed solution methodology are explained in section 4 and 5.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

This thesis is an action research on regression test-selection problem of an industrial project. In subject project manual regression test-selection method is used based on test team experience and feelings. Cost and value objectives of test cases are not taken into account in that method and full coverage of requirements affected from modification in the SUT cannot be provided.

Proposed regression test-selection methods in literature are reviewed and because of the source code access limitation and existence of many objectives other than source code- related ones, Multi-objective regression test optimization approach proposed by Harman [13] is found suitable for subject project.

It is structured by using a customized model of Multi-objective regression test optimization approach [13]. Used model for regression selection has two parts realized by using genetic algorithm: Requirement coverage based selection and Multi-objective minimization. First part of the model aims full coverage of requirements affected from modifications and it is realized with a constraint function that is applied each solution candidate emerged in genetic algorithm steps. In order to detect the test suites providing full coverage a traceability matrix that includes the connections between test case, high level requirements and low level requirements is used. This part improves the coverage of affected requirements of existing regression test selection approach.

Second phase of the model aims to find optimum test suite among the solution candidates that provides full coverage of affected requirements. In this phase ten different cost and value objectives are used to select optimum test suite. It is realized by using genetic algorithm which is a search based optimization algorithm. In order to be used objectives' data collected from project documents or created by stakeholders in fitness value calculations of genetic algorithm, a number of transformation processes are applied to the data. Then all objectives' data is scaled between 0 and 1 to be used in transformation of Multi-objective structure to single objective structure by using a weighting table. This table is created by stakeholders by using Delphi method. This phase of model improves the effectiveness of regression selection in test suite size and fitness value perspective.

Based on real industrial needs, this thesis has tackled a real life regression selection problem with several conflicting objectives by using MORTO. Application model of MORTO provides a general and flexible structure which is applicable to most of the regression test selection problems by customizing the structure based on specific data (objectives, weighting table, etc.) or tuning process (parameter tuning) of the problems. Basic contributions of this study are:

- A multi-objective formulation of the regression test selection problem by using ten different objectives.

- A customized and tuned genetic algorithm that provide full coverage of the requirements and multi objective minimization of test suites.

- Empirically evaluation of tailored multi-objective regression test optimization on an industrial project with specifically optimized genetic algorithm.

In scope of this thesis, two research questions are answered. The first one is "How can the GA be best calibrated to yield the best results? (RQ1)". In order to answer this question, basic structure of GA is analyzed and three important parameters that affect the performance of GA are determined: population size, crossover rate and mutation rate. GA was executed several times with different values of these parameters and results are compared in scope of execution time and fitness value. Based on this comparison the most efficient parameters values are determined and used in this study.

The second question is "How much cost improvement does the multi-objective approach provide compared to the previous existing test selection process? (RQ2)". In order to answer this research question, multi-objective regression test selection method and existing regression test selection process are compared based on test suite size, affected requirement coverage, irrelevant requirement coverage and fitness value perspective. Multi-objective regression test selection approach provides better solutions for all software versions than existing process in affected requirement coverage and fitness value perspective. Also in most cases it provides better solution than existing process in test suite size and irrelevant requirement coverage perspective. Because of the technical improbabilities, fault detection rate could not be added in this comparison. The reasons of this are explained in section 5.4. However, MORTO-proposed test cases are manually reviewed and for this specific case a diminishing risk is not detected.

This empirical study is an action research and it proposes a solution to a particular project. Objectives and parameters used in selection are project specific. In this perspective, a direct generalization of solution may not be meaningful. However, solution provides a general structure including two different phases: requirement coverage based selection and multi-objective minimization. This basic and flexible structure can be applied to most of the regression test selection problems by customizing the phases based on limits and constraints of the problems.

In customer organization, this solution will be applied to other projects with existing customization which includes objectives, weights and tuning values. Only project specific data will be collected and used in this structure.

6.2 Future Work

Future work of this thesis may be: (1) Expanding traceability matrix to include connection between source code and low level requirements: (2) Enhancement of genetic algorithm optimization by covering other parameters. (3) Generalizing the methodology to make it applicable for different regression test-selection problems.

Solution methodology proposed in this thesis includes a high level impact analysis by using a traceability matrix. That matrix does not include connection between low level requirements and source code. In future studies, connection between low level requirements and source code can be added to traceability matrix. In that circumstance, requirement coverage based selection phase of proposed solution methodology can be realized by using a source code analysis technique.

In Multi-objective minimization phase of this empirical study, population size and variation operators (crossover rate and mutation rate) of genetic algorithm are tuned with limited scale experimental comparisons. Other genetic algorithm parameters are not tuned in scope of this study. In future work, tuning of genetic algorithm can be extended to cover other parameters.

As defined in section 5.2, application model used in this thesis cannot be generalized with all parameters. In future studies, a more general version of application model can be structured.

REFERENCES

1. Gupta, R., M.J. Harrold, and M.L. Soffa, *An approach to regression testing using slicing.* Software Maintenance, 1992. Proceerdings., Conference on, 1992: p. 299-308.
2. Rothermel, G., *Efficient, Effective Regression Testing Using Safe Test Selection Techniques*. 1996, Clemson University: Clemson, SC, USA.
3. Rothermel, G. and M.J. Harrold, *A safe, efficient regression test selection technique.* ACM Transactions on Software Engineering and Methodology, 1997. **6**: p. 173-210.
4. Graves, T.L., et al., *An Empirical Study of Regression Test Selection Techniques.* ACM Transactions on Software Engineering and Methodology, 2001. **10**: p. 184-208.
5. Bible, J., G. Rothermel, and D.S. Rosenblum, *A Comparative Study of Coarse- and Fine-grained Safe Regression Test-selection Techniques.* ACM Trans. Softw. Eng. Methodol., 2001. **10**: p. 149-183.
6. Do, H. and G. Rothermel, *An Empirical Study of Regression Testing Techniques Incorporating Context and Lifetime Factors and Improved Cost-benefit Models*, in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2006, ACM: New York, NY, USA. p. 141-151.
7. Engström, E., M. Skoglund, and P. Runeson, *Empirical Evaluations of Regression Test Selection Techniques: A Systematic Review*, in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 2008, ACM: New York, NY, USA. p. 22-31.
8. Engström, E., P. Runeson, and M. Skoglund, *A systematic review on regression test selection techniques.* Information & Software Technology, 2010. **52**: p. 14-30.
9. Felderer, M. and E. Fourneret, *A Systematic Classification of Security Regression Testing Approaches.* Int. J. Softw. Tools Technol. Transf., 2015. **17**: p. 305-319.
10. Yoo, S. and M. Harman, *Pareto efficient multi-objective test case selection.* Proceedings of the 2007 international symposium on Software testing and analysis - ISSTA '07, 2007: p. 140.
11. Stringer, E.T., *Action Research*. 2007, Los Angeles: Sage Publications.
12. Coghlan, D. and T. Brannick, *Doing action research in your own organization*. 2010, Los Angeles: Calif. ; London: SAGE.
13. Harman, M., *Making the case for MORTO: Multi objective regression test optimization.* Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011, 2011: p. 111-114.
14. Gu, Q., B. Tang, and D. Chen, *Optimal Regression Testing based on Selective Coverage of Test Requirements.* 2010 International Symposium on Parallel and Distributed Processing with Applications (ISPA 2010). 2010: p. 419-426.

15. Gorschek, T., et al., *A Model for Technology Transfer in Practice.* IEEE Software, 2006. **23**(6): p. 88-95.

16. Sjoberg, D.I.K., T. Dyba, and M. Jorgensen, *The Future of Empirical Methods in Software Engineering Research*, in *2007 Future of Software Engineering*. 2007, IEEE Computer Society. p. 358-378.

17. Gay, L.R. and P.W. Airasian, *Educational Research: Competencies for Analysis and Application*. 2000: Merrill.

18. Garousi, V., *Empirical analysis of a genetic algorithm-based stress test technique*, in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. 2008, ACM: Atlanta, GA, USA. p. 1743-1750.

19. Pereira, G.C., M.M.F.D. Oliveira, and N.F.F. Ebecken, *Genetic Optimization of Artificial Neural Networks to Forecast Virioplankton Abundance from Cytometric Data.* 2013. **2013**: p. 57-66.

20. Yoo, S. and M. Harman, *Regression Testing Minimization, Selection and Prioritization: A Survey.* Softw. Test. Verif. Reliab., 2012. **22**: p. 67-120.

21. Catal, C. and D. Mishra, *Test case prioritization: a systematic mapping study.* Software Quality Journal, 2013. **21**(3): p. 445-478.

22. Singh, Y., *Systematic Literature Review on Regression Test PrioritizationTechniques.* Informatica 2012. **36**: p. 379–408.

23. Bates, S. and S. Horwitz, *Incremental program testing using program dependence graphs.* Proceedings of the 20th ACM SIGPLANSIGACT symposium on Principles of programming languages POPL 93, 1993. **5**: p. 384-396.

24. Wong, W.E., et al., *A Study of Effective Regression Testing in Practice* *. 1997: p. 264-274.

25. Jones, J.a. and M.J. Harrold, *Test-suite reduction and prioritization for modified condition/decision coverage.* IEEE International Conference on Software Maintenance, ICSM, 2001: p. 92-103.

26. Rothermel, G., R.H. Untch, and M.J. Harrold, *Prioritizing Test Cases For Regression Testing.* IEEE Transactions on Software Engineering, 2001. **27**: p. 929-948.

27. Aggrawal, K.K., Y. Singh, and A. Kaur, *Code coverage based technique for prioritizing test cases for regression testing.* ACM SIGSOFT Software Engineering Notes, 2004. **29**: p. 1.

28. Chen, Y., R.L. Probert, and H. Ural, *Model-based Regression Test Suite Generation Using Dependence Analysis.* Transition, 2007: p. 54-62.

29. Akhin, M. and V. Itsykson, *A regression test selection technique based on incremental dynamic analysis.* Software Engineering Conference in Russia (CEE-SECR), 2009 5th Central and Eastern European, 2009: p. 19-24.

30. Harrold, M.J. and M.L. Souffa, *An incremental approach to unit testing during maintenance*, in *Software Maintenance, 1988., Proceedings of the Conference on*. 1988. p. 362-367.

31. Orso, A., T. Apiwattanapong, and M.J. Harrold, *Leveraging Field Data for Impact Analysis and Regression Testing.* SIGSOFT Softw. Eng. Notes, 2003. **28**: p. 128-137.

32. Vokolos, F.I. and P.G. Frankl, *Empirical Evaluation of the Textual Differencing Regression Testing Technique Outsource Laboratories.* ACM Sigsoft International Conference on Software Testing and Analysis - ISSTA, 1998: p. 44-53.

33. Chittimalli, P.K. and M.J. Harrold, *Regression Test Selection on System Requirements*, in *Proceedings of the 1st India Software Engineering Conference*. 2008, ACM: New York, NY, USA. p. 87-96.

34. Krishnamoorthi, R. and S.A. Sahaaya Arul Mary, *Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases.* Inf. Softw. Technol., 2009. **51**: p. 799-808.

35. Srikanth, H., C. Hettiarachchi, and H. Do, *Requirements Based Test Prioritization Using Risk Factors.* Inf. Softw. Technol., 2016. **69**: p. 71-83.

36. Briand, L.C., Y. Labiche, and S. He, *Automating regression test selection based on UML designs.* Information and Software Technology, 2009. **51**: p. 16-30.

37. Farooq, Q.-u.-a., et al., *An approach for selective state machine based regression testing.* Proceedings of the 3rd international workshop on Advances in model-based testing - A-MOST '07, 2007: p. 44-52.

38. Gorthi, R.P., et al., *Specification-based approach to select regression test suite to validate changed software.* Neonatal, Paediatric and Child Health Nursing, 2008: p. 153-160.

39. Qu, B., et al., *Test Case Prioritization for Black Box Testing*, in *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01*. 2007, IEEE Computer Society: Washington, DC, USA. p. 465-474.

40. Chen, T.Y. and M.F. Lau, *A new heuristic for test suite reduction.* Information and Software Technology, 1998. **40**: p. 347-354.

41. Fazlalizadeh, Y., et al., *Incorporating Historical Test Case Performance Data and Resource Constraints into Test Case Prioritization*, in *Proceedings of the 3rd International Conference on Tests and Proofs*. 2009, Springer-Verlag: Berlin, Heidelberg. p. 43-57.

42. Khalilian, A., M. Abdollahi Azgomi, and Y. Fazlalizadeh, *An Improved Method for Test Case Prioritization by Incorporating Historical Test Case Data.* Sci. Comput. Program., 2012. **78**: p. 93-116.

43. Srikanth, H., L. Williams, and J. Osborne, *System test case prioritization of new and regression test cases*, in *Empirical Software Engineering, 2005. 2005 International Symposium on*. 2005. p. 10 pp.-.

44. Epitropakis, M.G., M. Harman, and E.K. Burke, *Empirical Evaluation of Pareto Efficient Multi-objective Regression Test Case Prioritisation.* 2015.

45. Yoo, S., R. Nilsson, and M. Harman, *Faster Fault Finding at Google Using Multi Objective Regression Test Optimisation.* Fse, 2011.

46. Pinto, G.H.L. and S.R. Vergilio. *A Multi-Objective Genetic Algorithm to Test Data Generation*. in *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*. 2010.

47. Bozkurt, M., M. Harman, and Y. Hassoun, *Testing web services: a survey*. 2010, Technical Report TR-10-01, Department of Computer Science, King's College London.

48. Ekelund, E.D. and E. Engstr, *Efficient Regression Testing Based on Test History : An Industrial Evaluation.* 2015: p. 449-457.

49. Engström, E., P. Runeson, and A. Ljung, *Improving regression testing transparency and efficiency with history-based prioritization - An industrial case study.* Proceedings - 4th

IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011, 2011: p. 367-376.

50. *Multi-objective optimization*, in *Wikipedia*. 2016.

51. *Heuristic (computer science)*, in *Wikipedia*. 2016.

52. Garousi, V., *Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms.* 2006.

53. Harman, M., et al., *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*, in *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*, B. Meyer and M. Nordio, Editors. 2012, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 1-59.

54. Gotshall, S. and B. Rylander, *Optimal population size and the genetic algorithm.* Proceedings On Genetic And Evolutionary Computation Conference, 2000: p. 1-5.

55. Arcuri, A. and G. Fraser, *On parameter tuning in search based software engineering*, in *Proceedings of the Third international conference on Search based software engineering*. 2011, Springer-Verlag: Szeged, Hungary. p. 33-47.

56. Spears, W.M. and K.A. De Jong, *An Analysis of Multi-Point Crossover*. 1995.

57. Patil, V.P. and D.D. Pawar, *THE OPTIMAL CROSSOVER OR MUTATION RATES IN GENETIC ALGORITHM : A REVIEW.* International Journal of Applied Engineering and Technology ISSN, 2015. **5**: p. 38-41.

58. Basili, V.R., G. Caldiera, and H.D. Rombach, *The goal question metric approach.* Encyclopedia of Software Engineering, 1994. **2**: p. 528-532.

59. Bohner, S.A. and R.S. Arnold, *Software Change Impact Analysis.* Aviation, 1996: p. 392.

60. Kim, J.-M.K.J.-M. and a. Porter, *A history-based test prioritization technique for regression testing in resource constrained environments.* Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, 2002: p. 119-129.

61. Eiben, A.E. and J.E. Smith, *What is an Evolutionary Algorithm?* Introduction to Evolutionary Computing, 2003: p. 15-35.

62. DeJong, K., *Learning with genetic algorithms: an overview*. Mach. Lang., 1988. **3**(2-3): p. 121-138.

63. Cobb, H.G. and J.J. Grefenstette, *Genetic Algorithms for Tracking Changing Environments*, in *Proceedings of the 5th International Conference on Genetic Algorithms*. 1993, Morgan Kaufmann Publishers Inc. p. 523-530.

64. Schaffer, J.D., et al., *A study of control parameters affecting online performance of genetic algorithms for function optimization*, in *Proceedings of the third international conference on Genetic algorithms*. 1989, Morgan Kaufmann Publishers Inc.: George Mason University, USA. p. 51-60.

65. M, H., #252, and hlenbein, *Parallel Genetic Algorithms Population Genetics and Combinatorial Optimization*, in *Proceedings of the 3rd International Conference on Genetic Algorithms*. 1989, Morgan Kaufmann Publishers Inc. p. 416-421.

66. C. Wohlin, P.R., M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* 2000.

67. Barros, M.d.O. and A.C.D. Neto, *Threats to Validity in Search-based Software Engineering Empirical Studies.* Relatórios Técnicos do DIA/UNIRIO, No. 0006/2011, 2011.

APPENDIX-1 GENETIC ALGORITHM PARAMETERS

#	Parameter	Value
1	PopulationType	'bitstring'
2	PopInitRange	<>
3	PopulationSize	30
4	EliteCount	'0.05*PopulationSize'
5	CrossoverFraction	0.7000
6	ParetoFraction	<>
7	MigrationDirection	'forward'
8	MigrationInterval	20
9	MigrationFraction	0.2000
10	Generations	'120'
11	TimeLimit	Inf
12	FitnessLimit	-Inf
13	StallGenLimit	20
14	StallTest	'averageChange'
15	StallTimeLimit	Inf
16	TolFun	1.0000e-06
17	TolCon	1.0000e-03
18	InitialPopulation	<>
19	InitialScores	<>
20	NonlinConAlgorithm	'auglag'
21	InitialPenalty	10
22	PenaltyFactor	100
23	PlotInterval	1
24	CreationFcn	@th_selectiveCreation
25	FitnessScalingFcn	@fitscalingrank
26	SelectionFcn	@selectionstochunif
27	CrossoverFcn	@th_crossover

28	MutationFcn	{<@th_mutate> <9/54>}
29	DistanceMeasureFcn	<>
30	HybridFcn	<>
31	Display	'off'
32	PlotFcns	<>
33	OutputFcns	<>
34	Vectorized	'off'
35	UseParallel	0