FACULTY OF ENGINEERING AND TECHNOLOGY
MASTER OF SOFTWARE ENGINEERING

Human Resource Optimization for Bug Fixing Planning Using Multi-Objective Evolutionary Algorithms

Author: Elias Khalil

Supervisor: Dr. Abdel Salam Sayyad

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Software Engineering at Birzeit University, Palestine

January 7, 2019

Approved by the thesis committee:

Dr. Abdel Salam Sayyad, Birzeit University

Dr. Yousef Hassouneh, Birzeit University

Dr. Samer Zein, Birzeit University

Date approved:

Declaration of Authorship

I, Elias Khalil, declare that this thesis titled, "Human Resource Optimization for Bug Fixing Planning Using Multi-Objective Evolutionary Algorithms" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at Birzeit University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

In software development projects, bugs are usually accumulated, and technical debt gets more significant over time. Managers decide to reduce the technical debt by planning one or more iterations for bug fixing. The time required to fix a bug depends on the bugs related competency areas, the human resource skill level in the assigned bugs components in addition to resource availability and dependency between bugs. Managers seek to achieve fixing the highest number of bugs during an iteration while at the same time fixing the highest possible number of high severity and high priority bugs while making sure that the time left to finish all bugs is minimal and on the other hand making sure that bugs are not starved in the backlog.

This research provides a framework to optimize human resource assignment to achieve the objectives above. First, it compares different many-objective evolutionary algorithms. Second, it measures the minimum run time to get near-optimal. Finally, validating the need for such framework by comparing human solutions and those generated by the framework on building a bug fixing plan.

This thesis uses various data sources by all parts which add more credibility to the results of experiments conducted. Data used varies from open source projects to industrial projects and of different sizes.

ملخص

في مشاريع تطوير البرمجيات ، عادة ما تكون قد تراكمت الاخطاء التقنيه وبالتالي يزداد مخزون هذه الاخطاء التي لم يتم معالجتها مع مرور الوقت. في مرحله معينه يقرر المديرون التقليل من الدين التقني عن طريق التخطيط لواحده أو أكثر من التكرارات لتصحيح الأخطاء.

يعتمد الوقت المطلوب لإصلاح الخلل على عوامل مثل الكفاءة ذات الصلة بالخلل ، ومستوى مهارة الموارد البشرية وطبيعة الخلل بالإضافة إلى توفر الموارد والتبعية بين الاخطاء. يسعى المدراء إلى إصلاح أكبر عدد من الأخطاء أثناء التكرار وفي نفس الوقت إصلاح أعلى عدد ممكن من الأخطاء ذات الأولوية العالية مع التأكد من أن الوقت المتبقي لإنهاء جميع الأخطاء اقل ما يمكن وعلى الجانب الآخر التأكد انه لا يتم بقاء الخلل بدون حل لفترة طويله .

يوفر هذا البحث إطار عمل لتحسين تخصيص الموارد البشرية لتحقيق الأهداف المذكورة أعلاه. أولا يقارن بين مختلف الخوارزميات التطورية متعددة الأهداف. ثانيًا ، يقيس الحد الأدنى لوقت التشغيل للحصول على الحد الأقصى الأمثل. وأخيرا ، التحقق من الحاجة إلى هذا الإطار من خلال مقارنة بين الإنسان والإطار على بناء خطة عمل لتصحيح الأخطاء

تستخدم هذه الرسالة مصادر مختلفة للبيانات من جميع الأجزاء التي تضيف مصداقية أكبر لنتائج التجارب التي أجريت. تختلف البيانات المستخدمة من مشاريع مفتوحة المصدر إلى مشاريع صناعية وبأحجام مختلفة.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my family especially for my wife Mary for her unlimited support during the past years. A special thank to my kids Miral, Madeleine and Marcel as the master time was taken from their family time.

I would like to express my sincere thanks to my advisor Dr. Abdel Salam Sayyad for his helpful guidelines, instructions, and advice.

Special thanks to all instructors of the software engineering master at Birzeit university whom I had a chance to enjoy and benefit from their lectures and efforts.

Contents

1 Introduction **1**

1.1 Problem Statement 2

1.2 Research Questions 4

1.3 Research Contribution 8

1.3.1 Automated Bug Fixing Planning Framework 8

1.3.2 Validating Framework on Three Different Datasets 9

1.3.3 Addressing the Problem Through Many objectives 10

1.3.4 Comparison Between Algorithms for Allocation Domain 10

1.3.5 Comparing NSGA-II with Human Performance (Study Validation) 11

1.4 Research Overview 12

1.5 Research Activities 13

2 Related Work **15**

2.1 Traditional Bug Assignment Approaches 16

2.1.1 Automatic Assignment 16

2.1.2 Semiautomatic Assignment 18

2.2 Search Based Bug Assignment Approaches 19
2.3 Thesis Distinction from Other Studies 22
2.4 Literature Review Summary 24

3 Background 26

3.1 Multi-objective Evolutionary Algorithms (MOEAs) and Pareto-front solutions 27
3.1.1 MOEAs applied in research 29
3.2 Evolutionary Genetic Algorithms 37
3.2.1 eGA general Design 38
3.2.2 eGA Operators 40
3.3 Evaluating multi-objective algorithms and HyperVolume (HV) 42
3.3.1 Hyper Volume metrics 43
3.4 JMetal- framework for developing metaheuristics for multi-objective optimization problems 45

4 Research Methodology And Experiment Setup 47

4.1 Experiment Data Sources 48
4.1.1 Eclipse Project Data 49
4.1.2 Industrial Data 52
4.1.3 Datasets Available Bug Properties 53
4.2 Experiment Setup 53
4.2.1 Chromosome Structure 53
4.2.1.1 Chromosome Operations 54

4.2.2 Multi-Objective Fitness Evaluation 56
4.2.3 Algorithms and jMetal Study 60
4.2.4 Human vs. Algorithm Setup 61
4.3 Experiment Assumptions 65

5 Experiments Results and Analysis 66
5.1 Algorithm Comparison 66
5.1.1 Algorithm Minimum Run Time 72
5.1.2 Human vs. SBSE Experiment 76
5.2 Framework Planner Advisor UI Tools 81
5.2.1 Solutions Radar Charts 81
5.2.2 Gannt Plan Chart 84
5.3 Results Impact and Use 85

6 Conclusion And Future Work 88
6.1 Threats to Validity 88
6.2 Conclusion 89
6.3 Difficulties and Obstacles 90
6.4 Future Work 91
6.4.1 Man in the loop support 91
6.4.2 Integration with Management tools 93

Bibliography 94
.1 Appendix A : Thesis External Links 103

List of Figures

2.1 Sample Gannt Chart Produced by Framework 24

3.1 IBEA Pseudo Code [42] 32
3.2 NSGA-II Algorithm Procedure [2] 33
3.3 Cellular Genetic Algorithm (cGA) [34] 36
3.4 MoCell Algorithm pseudo code [34] 37
3.5 Basic GA algorithm steps [43] 39
3.6 HV equation [2] 44
3.7 Visualization for the HV matrix [9] 44

4.1 Experiment Chromosome Structure 54
4.2 Experiment UI Tool 62

5.1 HV Quality Indicators for JDTMilestoneM2 69
5.2 HV Quality Indicators for JDTMilestoneM3 69
5.3 HV Quality Indicators for JDTMilestoneM4 70
5.4 HV Quality Indicators for JDTMilestoneM5 70
5.5 HV Quality Indicators for Industrial Dataset 1 70
5.6 HV Quality Indicators for Industrial Dataset 2 71
5.7 Run Time vs HV for JDTMilestoneM2 74

5.8 Run Time vs HV for JDTMilestoneM4 74
5.9 Run Time vs HV for JDTMilestoneM5 75
5.10 Run Time vs HV for Industrial Dataset1 75
5.11 Run Time vs HV for Industrial dataset2 76
5.12 Total Bugs Vs Priority 2d Projection 78
5.13 Total Bugs Vs Time 2d Projection 78
5.14 Total Bugs Vs Aging 2d Projection 79
5.15 Priority Bugs Vs Aging Factor 2d Projection 79
5.16 Priority Bugs Vs Time left to Fix all bugs 2d Projection . . . 80
5.17 Time left to Fix all bugs Vs Aging Factor 2d Projection . . . 80
5.18 Solutions Radar Chart 82
5.19 Record 84
5.20 A Solution Gannt Chart 84

List of Tables

2.1 Literature Review Summary 25

4.1 Sample of Eclipse Developers Dataset 50
4.2 Sample of Eclipse Bugs Dataset 51
4.3 Study Datasets Properties 53
4.4 Study Objectives Variables 59
4.5 jMetal Experiment Configuration 60
4.6 Experiment Bugs Details 63
4.7 Experiment Developers Details 64

5.1 Experiment Datasets Details 67
5.2 HV. Mean values 68
5.3 Average Rankings of the algorithms 71
5.4 HV convergence overtime for diffirent datasets 73
5.5 Hypervolume for NSGA-II vs Human-made Paretos 77
5.6 Partial Sample of a jMetal FUN File 83

Chapter 1

Introduction

Software product quality has been taking increasing attention due to our dependency on software in addition to the dramatic increase in the software product size. Bugs are part of any software development or maintenance process of any software product. The number of reported bugs increases proportionally with the product size. Studies show that the bigger and more complex the product is the more it is defect pron. This relationship increases monotonically [29] leading to a significant number of defects.

To deliver a product with high quality within time and budget, it requires spending high effort on testing and maintenance [49, 8]. Statistics showed that 80% of development cost is spent on bug fixing activities [45]. Unfortunately, projects budget is usually limited where just 32% of software projects are completed on time and within budget [15]. Additionally, developers usually produce a high rate of bugs during the project life cycle [36, 5, 41, 14]. These facts raise the need to balance the effort spent on testing and maintenance and the limited schedule and budget.

1.1 Problem Statement

Agile software development promotes an evolutionary change how software development process is defined and handled. In Agile methodologies, the team aims to deliver a product in an iterative, incremental manner. This is achieved by iteration-based development where a group of predefined user stories is selected for implementation in an iteration period. By the end of the iteration, a delivery is passed to the customer. This delivery is expected to be working as defined in the related user stories. To ensure the quality of an iteration quality, testing is performed throughout the iteration. Bugs discovered during the iteration are fixed as soon as they are discovered [44]. This fixing approach is crucial due to the customer expect ion of the iteration output. Unfortunately, this is not always the case where iterations usually end up with additional bugs added to the system. these bugs are not fixed in the iteration period due to developers availability and time limitations. Moreover, additional bugs are discovered throughout the regression process or are reported by customers using the product in the field.

With each iteration, more bugs are accumulated in the bugs backlog. At a certain period of time in the release timeline, management decides to reduce the size of the bugs backlog significantly. This is achieved by dedicating one or more iterations for bug fixing where some developers are freed to work on bug fixing activities. Setting a plan for a bug fixing iteration is not trivial especially if the number of targeted bugs are high. Project managers have to decide which bugs should be fixed in this period and who will fix each bug. Prior studies show that selecting bugs to fix within a target planning period, and the criteria of assigning bugs to developers are significant factors affecting bug fixing time [26, 4, 30, 49]. Additionally, assigning a bug to the right human resource has a significant impact on the maintenance period or the amount or nature of the fixed bugs during this period.

Reported bugs have various properties such as severity, priority, estimated time to fix (ETA)... etc. Managers usually use these properties as inputs to construct a bug fixing iteration plan. Additionally, planning takes into account the required human resources and related skills. Due to the various properties affecting bug fixing planning decisions, managers spend high effort and long time making sure they make a balance between different goals [30, 5]. This problem is more visible and could be more critical in large-scale projects due to the need to keep control over the size of the bugs backlog and the priorities of different bugs in addition to their impact on the overall system behavior, quality and budget.

The primary goal of the iteration is to best handle the bugs in the backlog as possible. The first goal that raises is reducing the number of defects, but this is not an ultimate goal. To improve the quality of the product through the iteration bug fixing and to increase customers satisfaction, managers set other goals of the iteration such as targeting the high priority and severity bugs. This increases customers satisfaction, but on the other hand, customers would like to see their reported bugs fixed even if they are not critical. These goals are usually competing where doing good on one goal does not necessarily lead to a good achievement of the other goals. Considering all goals makes the iteration planning a complicated and time-consuming task. Resource allocation is considered an NP-Hard complex problem [11]. The complexity of this problem arises from the high number of combinations of possible allocation and the impact of the allocation on product development time, cost and quality [22], and overall project success [7].

1.2 Research Questions

In this study, different perspectives of bug fixing planning problem are addressed using search-based software engineering (SBSE). SBSE has proved its efficiency to provide close-to-optimal solutions in various areas of software engineering including project management and resource allocation [1]. The fact that resource allocation impacts the plan output raises the need to identify a close-to-optimal HR allocation using SBSE techniques.

This study provides a framework used for constructing an automated plan. This framework takes off the effort to answer the following questions which are usually raised upon bug fixing iteration planning sessions:

RQ1: What is the best set of bugs to be fixed in the iteration period

This study targets medium to large scale projects with a bugs backlog size that is large and hard to optimize manually. Selecting bugs manually is not an easy task where the purpose of a good plan is to achieve an optimized plan achieving the best possible for all objectives aimed to be achieved.

Building a plan handling the objectives manually is a hard problem and time-consuming. To increase the number of bugs fixed in an iteration, the right bug should be assigned to the right person. Deciding the right person is done through bug required skills match with the developer acquired skills. Additionally, the availability of the developer and workload impact the amount and type of bug fixes she can achieve. Moreover, fixing severe or high priority bugs usually consumes more time and requires more skills than fixing trivial bugs.

Additional properties besides the severity and priority are taken into account in the planning phase. For example, customer reported bugs are usually considered high priority to be provided to the customer. Such bugs may not be with a real high priority from the product perspective and most of them are are not severe as these bugs are discovered by the customer after the product has passed multiple levels of testing activities. Providing fixes for such bugs is crucial for customer satisfaction where the customer feels self worth and attention. Other customer reported bugs may be critical, but such bugs are already with high priority and taken into account in the optimization. One more important property is the creation date of the bug. With a high rate of incoming bugs, some bugs get starved and don't have the chance to be fixed and stay in the backlog for ages. In order to minimize this starvation issue, aging is provided as a minimization objective in this research. Aging is measured base on the creation time, so to minimize the starvation, managers should always tend to include old bugs in the iteration plan.

Deciding on the bugs to be fixed in an iteration is not enough, hence long-term objective is also be considered. This objective is minimizing the time required to finish the rest of the bugs. Deciding the bug set to be fixed in the iteration period impacts directly the long-term time objective.

RQ2: Who is the best developer to fix each bug in the iteration

A decision to assign a bug to a specific developer, not another has a direct impact on the plan optimization. Bug triaging is a challenging decision as which bug or software module requires a different set of skills to work on. On the other hand, studies show that developer skill can be divided on 20 different skill level on each development category [13]. Bugs should be assigned to developers based on their skill, availability, and productivity. Assigning a bug to the improper developer who does not have the bug required skills impacts the time required to fix it.

Usually, bug triaging is a manual decision without using any automated tools. The triaging decision is mainly based on the triager decision based on previous knowledge about developers capabilities. This could be useful and applicable in small projects with a controlled bugs backlog. This could be more challenging and tedious when it comes to bigger projects where a large number of bugs are reported every day [23].

The main idea about choosing the right developer is that assigning a bug to a developer with the right skill gets it fixed in a shorter time than assigning it to another developer with different skills. The time it takes a developer to fix a bug depends on their skills on the required skills. Defining a mathematical relationship between required skills and developer skill can predict the relative time a developer can take to fix a bug compared to another developer.

RQ3: What is an excellent approach to achieve the iteration goals

Iteration goals may vary from one management person to another. This depends on the manager perspective, manager position, release target and customer demand [37]. For example, A manager may look forward to reducing the total number of bugs in the backlog, while another manager may look for better quality of the product by eliminating or reducing the severe bugs. On the other hand, another Marketing oriented personnel may be interested more in customer satisfaction by fulfilling his requested bug fixes. Based on the various perspectives, many optimal solutions may be available. All solutions which are results of the suggested framework are optimal where it presents the best results of one goal or objective without hurting the rest of the objective. This framework provides the managers or triagers with a set of optimal possible plans with fitness factors of each plan objective in order to let them select one solution to go with.

Automated solutions are excellent and make life easier, but usually, managers have some considerations to take into account to improve the reasonability of the results based on their perspective. This framework gives the ability for intervention with the flow of the algorithm direction while running. This approach is called **man in the loop** where the framework user can direct the results in one direction. In evolutionary algorithms, this is imposing exploitation to the algorithm search domain. Usually, managers are not interested in extreme results on objectives since in a competing objectives environment, improving one objective affects one or more of the other objectives. Managers can use this feature to set boundaries for the search domain. This is done a late stage of the framework runtime to give it more chance for exploration which mostly results in better results [16].

1.3 Research Contribution

In addition to answering bug fixing planning questions, this study provides some contributions by filling some gaps that have not been touched before in this area. Additionally, this study provides a practical framework that can be used in the industry especially that it is going to be validated over data imported from two different companies repositories.

1.3.1 Automated Bug Fixing Planning Framework

Based on this research an automated planning framework is defined and implemented to handle the complexity of human resource allocation for bug fixing. This framework consists of the following components:

- **Parser**: Reading bugs data from a dataset. Dataset structure defines a set of CSV files imported from other systems. Most bug management systems do not comply with this structure. Hence an additional migration tool should be built to provide the framework with the right files structure and format. Bugs data read from the dataset are loaded into objects holding properties and methods for bugs, developers, components and projects
- **jMetal Executer**: jMetal Framework [18] is used to run evolutionary algorithms to handle bugs allocation optimization problem.
- **Solution Selector**: A tool used to provide Framework users with a set of optimal solutions to select from. This is used for the final stage or for the man in the loop intervention.

1.3.2 Validating Framework on Three Different Datasets

Most related studies experiment the human resource allocation for bug fixing planning on a single dataset. In most cases, these datasets are open source datasets. An open source is still a valid option but it does not have all the required properties such customer input. Additionally, in open source projects, there is not dedicated developers, and developers select bugs to fix based on their convenience.

In this research, the datasets selected for the experiment are an Eclipse open source data set used by Karim [27] but using different objectives that what they have done. Additionally, this research targets two different datasets imported from real industrial bug repositories. These datasets contain multiple projects and multiple components per project.

1.3.3 Addressing the Problem Through Many objectives

All studies targeting the human resource bug fixing allocation problem that is conducted before this study consider a one or two objectives approach. This study approach takes a different dimension by increasing the number of objectives. This research is the first research to target many objectives instead of single and multiple objectives.

Through this research, the need and advantage of many objectives solution are presented providing the decision maker with more metric to choose one solution out of the optimal solutions set. Additionally, this research shows how adding more objectives does not hurt the solution space, while it provides more alternatives.

1.3.4 Comparison Between Algorithms for Allocation Domain

Many SBSE evolutionary algorithms are used to handle many software engineering problems. Different algorithms perform differently in a different domain. Bug allocation problem has some specialties such as skills, bug properties and limited time. This study conducts all experiments on more than one algorithms

The initial plan is to use NSGA-II, MoCell and IBEA algorithms for this research. These algorithms are selected due to their popularity in the SBSE area. The experiments don't include multiple parameter settings for each algorithm because this leads to a big range of confusing results especially that they are run on different projects too.

1.3.5 Comparing NSGA-II with Human Performance (Study Validation)

In order to show the power of SBSE for automatic building of a bug fixing plan. An experiment was conducted to compare human-built plans with plans (solutions) provided by NSGA-II algorithm. This experiment is run for 60 bugs backlog to build a one week plan. 31 volunteers participated in this experiment in one hour sessions. This experiment shows that NSGA-II solutions outperform human-made solutions. This is done by collecting all man-made solutions in a Pareto of non-dominated solutions. These solutions are compared with a Pareto of NSGA-II solutions using Hypervolume and by calculating the Euclidean distance between solutions in the two sets. Results of this experiment show the complexity of handling multiple objectives in a plan manually. Although this experiment was conducted on a relatively small dataset, it clearly presents the effectiveness of SBSE approach.

1.4 Research Overview

Intensive literature review has been conducted in order to find out the related work in the resource allocation and bug fixing planning fields. Additionally, focus in the literature review is put on SBSE and evolutionary algorithms. The rest of the research has been divided into the following chapters:

- **Related Work**: Intensive literature review investigating the studies conducted on both human resource allocation and bug fixing planning. Additionally, studies using SBSE are highlighted and is compared to this study methodology and contribution.
- **Background**: A description of the theory behind this study and the tools used. Meta-heuristic optimization is explained in the context of this thesis study describing non-dominated solutions and their relation to the solutions that can be provided for the decision makers.
- **Research Methodology And Experiment Setup**: Describing the different dataset used in this study in addition to the setup used. This includes the algorithms configuration, bugs and developers data in addition to human validation experiment setup.
- **Experiments Results and Analysis**: presenting the results of the experiments done and analyzing them. This includes deciding of

the best algorithm to be used and the time required to run it. Additionally, this section analyzes the human vs. algorithm experiment to present the power of solving this optimization problem through SBSE

- **Conclusion**: summarizing the results of this study while presenting the threats to validity. Future work is suggested in this chapter based on this study results.

1.5 Research Activities

Target research includes three different datasets, and each dataset contains one or more project in addition to various sizes of developers available for study. The following activities are conducted:

- **Detailed jMetal experiment**: A jMetal experiment is built to present the objectives and approach and get result presenting the importance of the study and problems it targets. This is achieved by parsing the experiment data in a structure consuable by jMtal framework, constructing and tuning jMetal operators, then get optimal results from the jMetal.

- **Algorithms Comparison**: Three algorithms are used in this study (IBEA, MoCell, and NSGA-II). Quality of the Pareto optimal solutions output of each algorithm is measured on each dataset and compared.

- **Algorithm minimal run time**: Finding the minimal time required to find a set of solutions near to optimal.
- **Algorithm vs. Human plans**: To present the power of the suggested framework, Managers are asked to provide manual planning for one or more dataset to be compared by the framework output.

Chapter 2

Related Work

Human resource allocation for bug fixing planning is an old problem that has been frequently approached in the literature review due to the importance of this problem. Bug fixing is one of the important topics in software testing and maintenance [24]. Many studies approached the issue of bug fix planning [14, 47, 49]. Most of those studies used traditional methodologies to address this problem.

In this chapter, a comprehensive review will be presented for previous studies that have approached this problem. This review is classified into two categories. Traditional approaches targeting this problem is the first category where researchers have used different automatic algorithms to decide the best-optimized bug assignment. The second category includes researches that have adopted search-based software engineering (SBSE) and meta-heuristic

2.1 Traditional Bug Assignment Approaches

Bug assignment has been studied widely in the past decade. Literature review divides the traditional assignment approaches into automatic and semi-automatic approaches. Different types of algorithms are used in the automatic assignment for bugs to developers where it does not require human intervention. On the other hand, semiautomatic approaches provide humans with parameters and recommendations to help humans to take assignment decisions.

2.1.1 Automatic Assignment

Cubranic and Murphy have propose an automated solution for bug triage [33]. They apply machine learning techniques to help decision makers to triage bugs. This study uses text categorization to decide who is the best developer to work on a bug. Eclipse project is used to present the contribution and capabilities of this study using a prototype application running on 15,859 bugs. The main idea behind this study was to suggest an approach which assigning a new reported bug to a developer without any human intervention and any change of the assignment or fixing process and tools.

The framework used by Cubranic and Murphy is based on the problem of assigning a text document into one or more topic categories or classes based on a study done by McCallum [32]. Each developer is mapped to one class while a bug is mapped to a document representing a supervised learning problem. Many supervised learning problem can be used for text classification such as k-nearest neighbor and decision trees. The approach determines the best suitable developer to assign bug two. This does not guarantee the best optimization including cost, developer load, and timing.

Latent Semantic Indexing is another approach used for automatic bug triage. A study conducted by Ahsan and others [3] has used techniques based on bug categorization. 1,983 resolved bug are selected for this case study where the suggested framework is used to triage bugs to developers then comparing the results to the already resolved bugs. The algorithm has achieved 44% classification accuracy. Results obtained through this framework are interesting but still, it does not take any other objectives in the triage process.

Tossing graphs is another approach used for bug triage in lecture. Both Jeong [**t**]jeong2009improving and Bhattacharya [10] have used this approach. Both studies have used Mozilla and Eclipse bugs and applied Tossing graphs to get the best developer to be assigned to a bug. Tossing graphs uses classifiers (Markov-model based) to recommend potential developers. Related studies use fine-grained updated and added extra attributes for classification; using additional attributes on edges and nodes while explaining the importance of adding additional attributes and their impact on the results and accuracy.

Training set reduction is another approach used for bug triage by Zou and others [52]. In this study, a training set reduction is proposed using the combination of feature selection and instance selection. Their training set approach is a process consisted of 2 phases which includes feature selection which is responsible for removing irrelevant words and instance selection which is responsible for removing irrelevant bug reports. This study was applied on Eclipse project.

2.1.2 Semiautomatic Assignment

Anvik et al. [5] presented an approach for semi-automating the assignment of bug to a developer. They applied a machine learning algorithm based on text categorization. Their approach uses a bug repository; the data form this repository is used later on by an algorithm to predict who can fix each bug. They solved problems with large numbers of bug. However, their approach does not address the bug fixing time, and bug assignment issue. This study also uses Eclipse bugs for the experiment.

Severity is also suggested in addition to bug triage in a study done by Yang and others [48]. This study is essential due to addressing a severity factor in addition to a bug to developer assignment. The topic of the bug is extracted depending on historical information taken from the bugs repository for fixed bugs to find similar bugs. New bugs are compared to previously categorized reports based on multiple features where severity of the bug is related to the features of bugs in the same topic. Datasets from Eclipse, Mozilla, and Netbeans are used to validate this approach.

2.2 Search Based Bug Assignment Approaches

In this section, studies which use SBSE to approach bug to developer assignment are reviewed. These studies have targeted the problem from some perspectives and objectives, but still, there is much space for this thesis to contribute.

Xiao and Afzal [47] used SBSE to address resource scheduling for bug fixing tasks problem. Their objective was to minimize schedule time that is required to fix a set of given bugs. The inputs for their study are Bug properties, Resource skills, and availability. In their work, they treated all factors as one objective with different constraints. Moreover, they proposed a multi-objective fitness function by linearly combining all the objective with different weights. This study took skills into account, but skills were mainly around components. For example, Database skill was required when working on database and UI skill is required when working on UI components. It did not address the cases where multiple skills are required to address a bug in a particular component.

Study of Xiao and Afzal [47] focused on supporting many objectives but getting deep into this study, it is apparent that it was not mean a meta-heuristic multi-objectives. Actually, it was weights added based on multi-objective to come up with a single objective solution based on GA and hill climbing algorithms.

This study addresses this problem using many-objective formulations, and finding sets of Pareto-efficient solutions. In addition, no fixed number of bugs to be solved is defined. Instead, it is treated as an objective to be achieved in a fixed period iteration. Because in reality developers usually have more bugs than what they can fix in a single iteration [5, 41].

Karim and Rahman have conducted two studies on the bug to developer assignment [27, 38]. In the first study [38] They have suggested a framework to assign one developer to each bug, while in the second study [27] they adopted an approach of assigning one or more developers to a single bug. This study is the main reference for our study due to the following similarities:

- Each bug has a set of skills called competency areas required to be acquired by a developer to be fixed. A set of competency areas are defined for each project and each bug a percentage value for each competency area is set indicating the ratio of work required per competency area.
- Part of this thesis experiment is using the same dataset used by Karim and Rahman. It uses Eclipse Platform and JDT but with different perspective and scope.
- Supporting multi-objective study. They have studied two objectives: time and cost. This thesis approach is also multi-objective, but they are different.

Another type of related work is task planning and human resource allocation using SBSE. Ren et al. [40] conducted an empirical study to optimize both developer team staffing and work package scheduling through cooperative co-evolution to achieve early overall completion time. They used a single fitness evaluation, and some factors such as developer skills and expertise are omitted. Data used in this study is imported from real industrial cases.

Park et al. [35] suggested a GA approach for solving human resource allocation problem reflecting some practical considerations. Their objectives were to finish a set of tasks by a set of developers with minimal time, least developer multitasking time, assignment on relevant tasks and balance of allocation. Their approach assumed finishing all tasks. However, in bug fixing, developers may not be able to fix all bugs due to large accumulated bugs backlog, and during the software life, new bugs are reported everyday [23].

Kang et al. [26] addressed constraint-based human resource allocation in the software project. They used accelerated simulated annealing (ASA) with constraints to achieve their objective to optimize scheduling of human resources allocation. Their study does not take task priorities into account, while our study does. Additionally, in this study, these constraints are treated as objectives, so a manager is given a set of various solutions to select from a Pareto front of solutions.

A study using multi-objective SBSE for tasks resource allocation is conducted by Bibi [12]. The three objectives are to increase resource utilization, decrease the cost of development and decrease Implementation time. They have made a comparison between three algorithms: GA, and multi-objective particle swarm (MOPSO) in addition to elicit non-dominated sorting evolutionary strategy. The approach followed by this study is using data from previously published cases study while comparing their experiment results with the published case study showing improvement in all objectives. MOPSO was the main contribution of this study where it outperformed the other algorithms. Data for the case study was custom data provide on an MS project.

2.3 Thesis Distinction from Other Studies

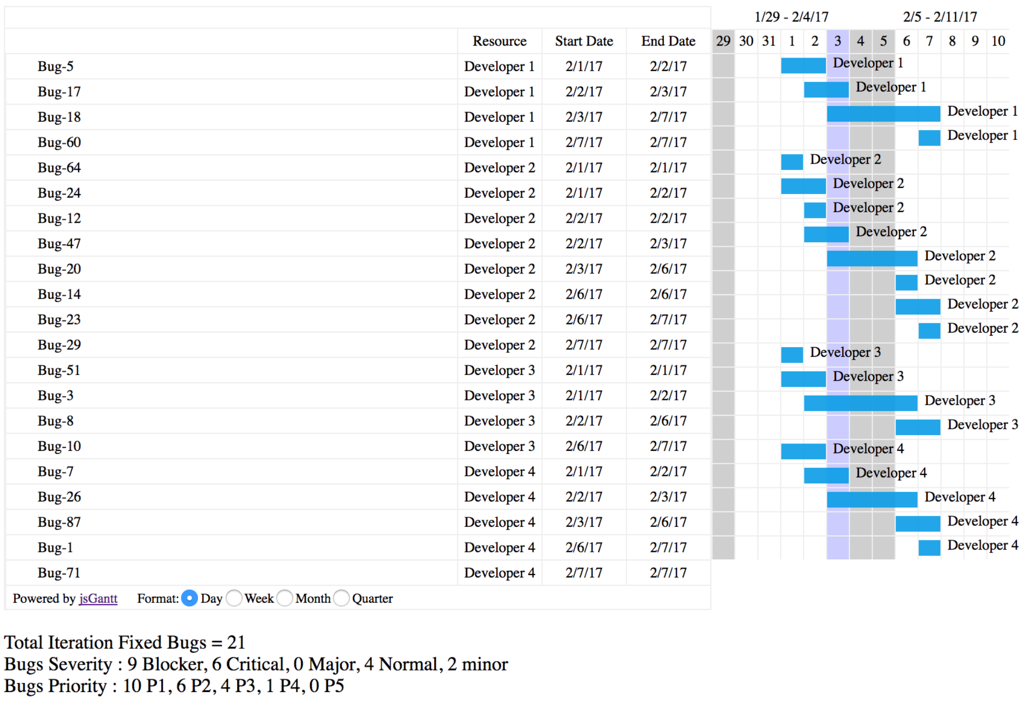
This thesis differs from the above-related work in some aspects which presents its contribution in the bug to developer allocation area. The differences can be summarized in the following points:

- Many-objective: All tradition studies have focused on bug triage more than building a plan. This did not include any other objective else than selecting the best fit developer. Some SBSE studies have targeted this problem but most of them are single objective studies. This thesis is targeting many objectives which have never been investigated in literature such as Customer priorities and bugs aging.

- Industrial data: human resource allocation for bug fixing is a real industrial problem especially when it comes to large projects. As described in Table 2.1 most studies uses open source projects bugs for the experiment. In industrial data, extra properties are included such as customer priorities and real targeted developers classifications. A previous study [28] shows that adding more objectives do not hurt the overall quality of the other objectives while it gives managers more insights into output solutions.

- Iteration based: Recently most development processes are adopting iterative, incremental methodologies. This means that a fix size team has a fixed period to accomplish specific objectives. Most studies focus on reducing time and cost while fixing all bugs. This is not practical as it is rare that a big project has all bugs fixed. This thesis consider a fixed size team which means a fixed cost in addition to fixed iteration time that is predefined and agreed on with project stakeholders which means a fixed time constraint. Hence, both objectives are removed from the problem, and other objectives are targeted.

- Simple and easy to integrate: This thesis sets design for an automated framework for planning. Inputs and outputs are easily translated into a real process related data. A simple interface can be integrated with bug repositories such as Jira or Rally to read bugs backlog, developers and timing and also framework can be integrated to automatically save a plan to the system based on human selection out of the Pareto optimal solutions coming out of the system. This framework is capable of generating plan Gannt charts as illustrated in figure 2.1 for human understanding of the generated solutions.



	Resource	Start Date	End Date
Bug-5	Developer 1	2/1/17	2/2/17
Bug-17	Developer 1	2/2/17	2/3/17
Bug-18	Developer 1	2/3/17	2/7/17
Bug-60	Developer 1	2/7/17	2/7/17
Bug-64	Developer 2	2/1/17	2/1/17
Bug-24	Developer 2	2/1/17	2/2/17
Bug-12	Developer 2	2/2/17	2/2/17
Bug-47	Developer 2	2/2/17	2/3/17
Bug-20	Developer 2	2/3/17	2/6/17
Bug-14	Developer 2	2/6/17	2/6/17
Bug-23	Developer 2	2/6/17	2/7/17
Bug-29	Developer 2	2/7/17	2/7/17
Bug-51	Developer 3	2/1/17	2/1/17
Bug-3	Developer 3	2/1/17	2/2/17
Bug-8	Developer 3	2/2/17	2/6/17
Bug-10	Developer 3	2/6/17	2/7/17
Bug-7	Developer 4	2/1/17	2/2/17
Bug-26	Developer 4	2/2/17	2/3/17
Bug-87	Developer 4	2/3/17	2/6/17
Bug-1	Developer 4	2/6/17	2/7/17
Bug-71	Developer 4	2/7/17	2/7/17

Powered by jsGantt Format: [✓ Day] Week Month Quarter

Total Iteration Fixed Bugs = 21
Bugs Severity : 9 Blocker, 6 Critical, 0 Major, 4 Normal, 2 minor
Bugs Priority : 10 P1, 6 P2, 4 P3, 1 P4, 0 P5

FIGURE 2.1: Sample Gannt Chart Produced by Framework

2.4 Literature Review Summary

Table 2.1 summarize related work done on bug to developers assignment problem. It includes both traditional and SBSE approaches. Traditional approaches are comprehensive while on the other hand SBSE studies are limited in number and contribution which raise the need for this study.

Study	Approach	Dataset	Open Source	# Obj
Automatic bug triage using text categorization [33]	Machine learning	Eclipse	yes	-
Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine [3]	Bug categorization	Mozilla	Yes	-
Improving bug triage with bug tossing graphs [25]	Tossing graphs	Mozilla & Eclipse	Yes	-
Automated, highly-accurate, bug assignment using machine learning and tossing graphs [10]	Machine learning and tossing graphs	Mozilla and Eclipse	Yes	-
Towards training set reduction for bug triage [52]	Training set reduction	Eclipse	Yes	-
Who should fix this bug? [5]	Machine learning	Eclipse	Yes	-
Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug report [48]	Historical information	Eclipse, Mozilla, and Netbeans	Yes	-
Search-based resource scheduling for bug fixing tasks [47]	SBSE - GA	Industrial	NO	1
An empirical investigation of a genetic algorithm for developer's assignment to bugs [38].	SBSE - GA, K-Greedy	Eclipse	Yes	1
An empirical investigation of single-objective and multi-objective evolutionary algorithms for developer's assignment to bugs [27]	SBSE - GA	Eclipse	Yes	2
Cooperative co-evolutionary optimization of software project staff assignments and job scheduling [40].	SBSE -CCEA	Industrial	No	1
Practical Human Resource Allocation in Software Projects Using Genetic Algorithm [35].	SBSE - GA	Custom	NO	1
Constraint-based human resource allocation in software projects [26].	Accelerated simulated annealing	Government information system	No	1
Comparison of Search-Based Software Engineering Algorithms for Resource Allocation Optimization [12].	SBSE- MOPSO	custom	No	3

TABLE 2.1: Literature Review Summary

Chapter 3

Background

Human resource allocation for bug fixing involves competing objectives such as a total number of fixed bugs in the iteration vs. number of fixed severe bugs in this iteration. Multi-objective metaheuristic algorithms are designed to get an optimal solution. With multi-objective, there is no single optimal solution but a set of optimal solutions where no other solution is better than in the search space. These solutions are called non dominated solutions. Multi-objective algorithms provide a Pareto front of non dominated solutions [42].

This chapter presents the theoretical background in details behind multi-objective problems and algorithms and the Pareto-front solutions, and focus on the algorithms used for this research. Evolutionary genetic algorithms theory is introduced in addition to operators used for algorithm's execution. Following that; HyperVolume metric which is used for evaluating the solutions obtained from the algorithms comparison is presented and discussed. Finally, a full description of the jMetal framework is discussed as it is used for problems evaluations

3.1 Multi-objective Evolutionary Algorithms (MOEAs) and Pareto-front solutions

A multi-objective optimization problem is considered a complex problem; as it states the challenge of evaluating more than two objectives in most of the times; and these objectives are contradicting and competing with each other, so the need for a computational algorithm is becoming higher.

Optimization problems are supposed to find the optimal solution from a considerable search space; which adds to the complexity of the problem; so finding the best and the optimal solution becomes nearly impossible using exact search methods and algorithms. The solution to this dilemma is to use metaheuristics algorithms, which provides a nearly optimal solution for complex problems. What adding to the complexity of the problem is that humans find it nearly impossible to perform the evaluation manually without the help of algorithms. The computational power and resources needed to solve the problem is high, and time to obtain the optimal solution is a critical factor in solving problems.

Evolutionary Algorithms are classified under metahueristic algorithms, and it is applied to solve multi-objective optimization problems by providing solutions near to the optimal solution as much as possible based on an evaluation factor. Over the years and since Since 1985 [51], algorithms were studied and applied to solve real-world software engineering problems. An important aspect of evolutionary algorithms is that no generalization can be adapted and say that one or set of algorithms outperforms others for all or most optimization problems [18]. Reasons can be summarized in four main points according to Durillo and Nebro in there study about jMetal [18], first is the absence of a benchmark for MOEAs that is internationally accepted and can be refer to, second is that no metrics agreed on to be used to evaluate the performance of the algorithm and then rating it according to results from other algorithms; third, the variation in the parameter settings used in different studied plays a role in making it harder to generalize conclusions, finally; different implementation of metaheuristics algorithms affects the numerical results and the performance of the algorithm based on the programming language for example.

Multi-objective problems present a set of solutions, which are referred to as the problem search space. The search space for a multi-objective problem has many solutions in comparison with the single objective problem that may have one unique optimal solution. Each solution in the search space is represented as a vector, the components of the vector are the value scored according to each objective, and later on used to perform the trade-off between these solutions, when the decision maker uses his implicit knowledge to compare between the alternatives and then selecting the most acceptable one for their related problem [51].

Each multi-objective problem consists of A Pareto-front is a method used to represent the solutions in a search space, as points on an N coordinates space, where each coordinate represents one of the objectives of the problem, a Non-dominant solution is: a solution where no other solution is better than it.

A solution $x^{(1)}$ is said to be dominating $x^{(2)}$ if $x^{(1)}$ is not worse than $x^{(2)}$ in all objectives and $x^{(1)}$ is better than $x^{(2)}$ in one or more objectives [31]. Measuring how much the objective is good is handled through defining fitness criteria for each objective which could be maximized or minimized for better results.

Many optimization search based algorithms have been studied over the past 20 years. Evolutionary algorithms have provided significant solutions for both single and multi-objectives search-based problem.

3.1.1 MOEAs applied in research

In this research we are using the following MOEAs:

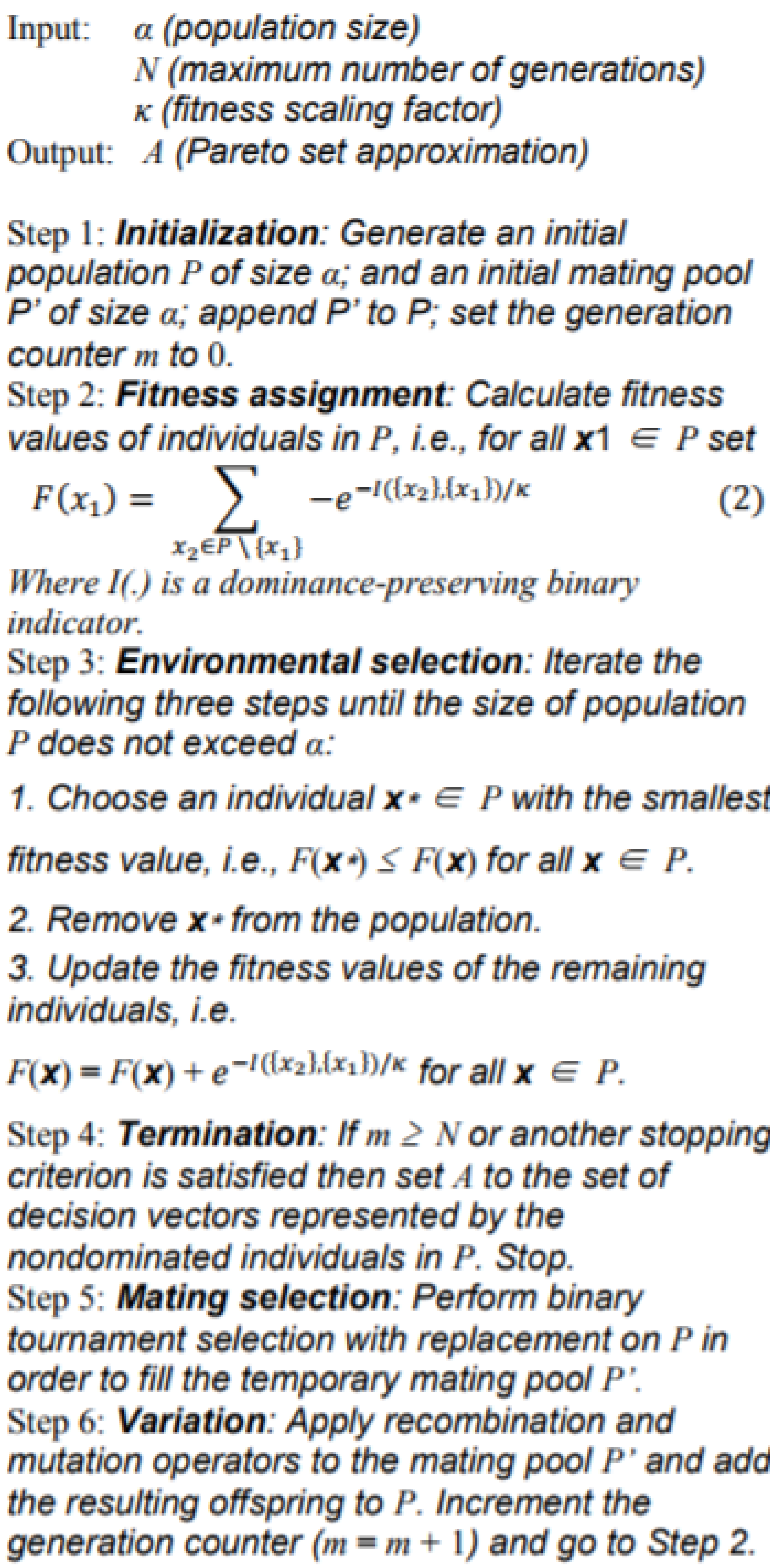
1. Indicator-Based Evolutionary Algorithm (IBEA)
   IBEA is one of multi-objective optimization algorithm that is a preference-based evolutionary algorithm [50]. The algorithm is able to capture the preference of the decision maker and use it to direct the search process for the multi-objective problem. The preference is being defined in the start of the search process and named the indicator, then make it the base of the execution and selection process afterward, this means that the preference guides the whole search process in a straightforward process without the need of other sharing operators or diversity techniques. Also, using IBEA the population size

can be both arbitrary and faster than other algorithms as it performs the comparison over pairs and not over the whole population set [50]. IBEA uses preference value to find Pareto-optimal solutions, so it achieves the most preferred solution [42].

The algorithm stands out in its dominance criteria to minimize the distance between the true and the obtained Pareto fronts and to maximize the diversity of the obtained Pareto-fronts. IBEA assigns to each solution a weight based on quality indicators, which applies that user preferences are given more wights and factoring [42]. Other algorithms apply to rank for each in the solution after being refined against information captured from the objective space; although different algorithms; other than IBEA; applies this in different aspects, still most of them were not able adapt to changes according to the preference-based in different solutions, with the lack of this flexibility one approach was applied to different problems and preferences, which led to the failure in capturing specific problem objectives in the obtained Pareto-fronts [50].

Figure 3.1 shows the algorithm pseudocode in reference to [42]. The main procedure of the algorithm starts with the generation of the population P of size n, and initializing the value of the counter m to be 0. The second step is the fitness assignment for the individuals in the obtained Pareto P. In the third step; an environmental selection is used by performing a loop that iterates over the individuals until the size of the population P does not exceed n by choosing the individual with the smallest fitness then removing it from the population P and do an update for the fitness values for the rest of the individuals. This step involves a fitness factor k. The fourth step is the termination step for the loop mentioned in step three, where a stop criterion is met so the vector of the individual in the decision variable now satisfies criteria. The fifth step performs a binary tournament selection with replacement in P, the mating used in this step fills the temporary population pool P'. The final step involves applying mutation and recombination operators to the pool P' and get offspring and adding them to P, and increasing the counter m by one, and go back to step two [50].

2. Non-dominated Sorting Genetic Algorithm II (NSGA-II)
   NSGA-II is an evolutionary algorithm that presented an algorithm that solved other previously algorithms drawbacks. The first issue was computational complexity which leads to more resources consumption during execution, NSAG-II presented a faster non-dominated sorting algorithm with complexity of $O(mN^{(2)})$ in comparison to previously proposed algorithms that has $O(mN^{(3)})$ complexity [17]. The second issue is that other MOEA algorithms; up to the time of developing NSGA-II; were criticized by the lack of non-elitism approaches which relates to higher computational complexity and lack of exploitation support [17]. The third issue is that other MOEAs needed to set up a specific value for the sharing parameter which is used in the sharing process to ensure variations



```
Input:  $\alpha$ (population size)
        $N$ (maximum number of generations)
        $\kappa$ (fitness scaling factor)
Output: $A$ (Pareto set approximation)

Step 1: **Initialization**: Generate an initial
population $P$ of size $\alpha$; and an initial mating pool
$P'$ of size $\alpha$; append $P'$ to $P$; set the generation
counter $m$ to 0.
Step 2: **Fitness assignment**: Calculate fitness
values of individuals in $P$, i.e., for all $\mathbf{x}1 \in P$ set
    $F(x_1) = \sum_{x_2 \in P \setminus \{x_1\}} -e^{-I(\{x_2\},\{x_1\})/\kappa}$    (2)
Where $I(.)$ is a dominance-preserving binary
indicator.
Step 3: **Environmental selection**: Iterate the
following three steps until the size of population
$P$ does not exceed $\alpha$:
    1. Choose an individual $\mathbf{x}* \in P$ with the smallest
    fitness value, i.e., $F(\mathbf{x}*) \leq F(\mathbf{x})$ for all $\mathbf{x} \in P$.
    2. Remove $\mathbf{x}*$ from the population.
    3. Update the fitness values of the remaining
    individuals, i.e.
    $F(\mathbf{x}) = F(\mathbf{x}) + e^{-I(\{x_2\},\{x_1\})/\kappa}$ for all $\mathbf{x} \in P$.
Step 4: **Termination**: If $m \geq N$ or another stopping
criterion is satisfied then set $A$ to the set of
decision vectors represented by the
nondominated individuals in $P$. Stop.
Step 5: **Mating selection**: Perform binary
tournament selection with replacement on $P$ in
order to fill the temporary mating pool $P'$.
Step 6: **Variation**: Apply recombination and
mutation operators to the mating pool $P'$ and add
the resulting offspring to $P$. Increment the
generation counter ($m = m + 1$) and go to Step 2.
```

FIGURE 3.1: IBEA Pseudo Code [42]

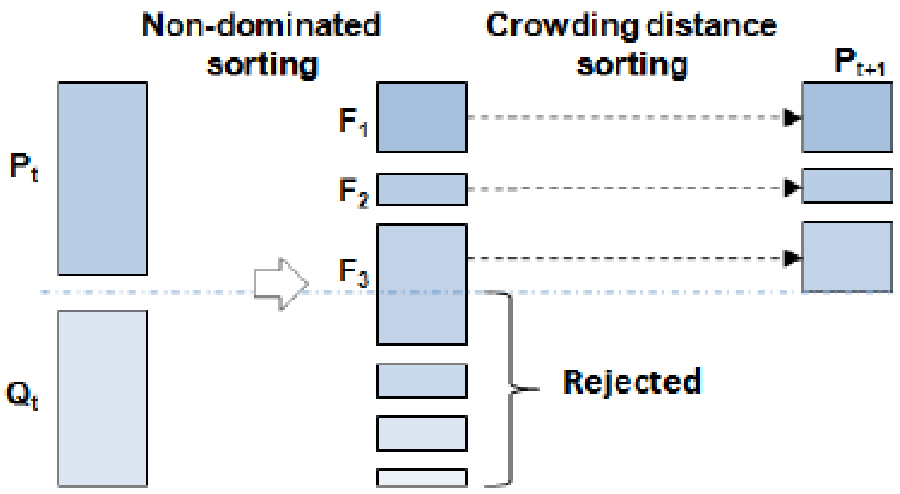


FIGURE 3.2: NSGA-II Algorithm Procedure [2]

in the obtained solutions, a parameterless approach is better [17] as the value of this parameter has to be assigned a given value and so bias in the obtained solutions. NSGA-II solved this parameterized issue by presenting a selection operator to select the best chromosome from parents/children pool in a population by referring to its fitness and spread values. The innovation in the multi-objective approach followed by NSGA-II added diversity and spread to the solutions obtained in the Pareto-front and made NSGA-II a remarkable algorithm with satisfying results if compared with other algorithms [17].

Figure 3.2 illustrated NSGA-II procedure in four steps. In the first step, a population R_t is randomly generated, this population includes parent population P_t and then applying operators including mutation and Binary tournament selection, recombination, to generate offspring population Q_t of the size N; chromosomes in the population R_t are assigned the fitness value that refers to its non-domination level, and they are being ranked and sorted based on non-domination where minimizing the fitness value is desired. The result of this sorting process will produce fronts F_i where $i = 1, 2, 3, ...$ In the second and third steps, a new population $P_(t + 1)$ is be being set from the best F ranked and passed to the next step; if the size of F_1 is less than N then the chromosomes are being selected according to the crowding distance approach by measuring the density of the solution from its neighbors and then comparing values between all solutions, and passed to the next step, and rest of the solutions in $F_2, 3, ...$ are rejected. If the not, then the population includes chromosomes from $F_2, 3, ...$ and so on. Finally in the fourth step; Operators including crowded tournament selection, crossover and mutation will be applied to get a new population $Q_(t - 1)$ form $P_(t + 1)$. In each iteration, a counter i is increased by 1 as the [2] [17].

3. Multi-Objective Cellular Genetic Algorithm (MoCell)

   MoCell main characteristic is the usage of an external archive to keep non-dominated solutions. The elements of the archive are used for feedback where individuals in the population are randomly replaced in each iteration so elite solutions are included in the obtained front with higher probability [34].

   The MoCell algorithm is built based on Cellular Genetic Algorithms (cGA). Figure 3.3 illustrates the cGA; where the algorithm structure the population and apply mutation and crossover operators on parts of the population instead of applying operators to the population as a whole. MoCell applies the cellular concept where a neighborhood is highly used; so in each breading loop of the algorithm, each individual in the population only contact with its neighbors. This increases the exploration in the search space and exploitation is assured by applying operators inside neighborhoods. This mechanism makes better sampling in the search space which increases the quality of the solutions obtained in most cases [34].

Figure 3.4 shows the steps of the MoCell algorithm. The algorithm starts with preparing an archive which is empty to store non-dominated solutions which are named the Pareto-front. The individuals in the population are divided into a two-dimensional grid, where a breeding loop is active to apply genetic operators over them until a termination condition is met. In this loop, every two individuals are selected from the neighbors, apply crossover and mutation to generate the offspring, this offspring is being evaluated against the fitness function. After that, the offspring is compared with the individuals in the auxiliary population and if its fitness value is better than individual in the current position and if not it is ignored, and in case both are nondominated then the individual with better crowding distance measure takes place at the current position. Another

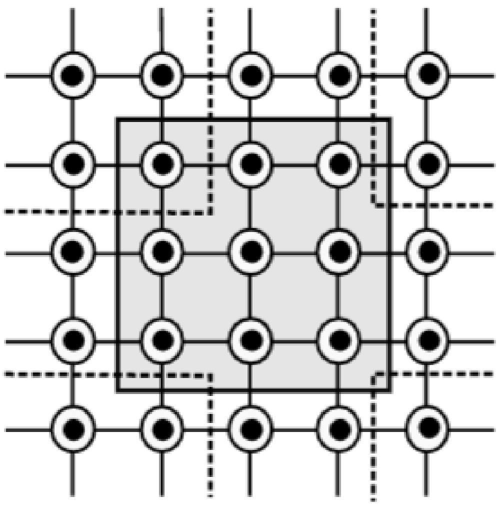


FIGURE 3.3: Cellular Genetic Algorithm (cGA) [34]

comparison to be inserted in the external archive; given that all individuals in this archive are ranked according to the crowding distance measure, then the individual is inserted, and if the archive is full, then the individual with the worst crowding distance measure is removed. The final step of the algorithm includes replacing the old population with the auxiliary population, which means better individuals are passed to the next generation, also feedback is performed to replace individuals with random size from the external archive; i.e the Pareto front with individuals from the population [34].

```
proc Steps_Up(mocell)        //Algorithm parameters in 'mocell'
Pareto_front = Create_Front() //Creates an empty Pareto front
while !TerminationCondition() do
   for individual ← 1 to mocell.popSize do
      n_list←Get_Neighborhood(mocell,position(individual));
      parents←Selection(n_list);
      offspring←Recombination(mocell.Pc,parents);
      offspring←Mutation(mocell.Pm,offspring);
      Evaluate_Fitness(offspring);
      Insert(position(individual),offspring,mocell,aux_pop);
      Insert_Pareto_Front(individual);
   end for
   mocell.pop←aux_pop;
   mocell.pop←Feedback(mocell,ParetoFront);
end while
end proc Steps_Up;
```

FIGURE 3.4: MoCell Algorithm pseudo code [34]

3.2 Evolutionary Genetic Algorithms

In the past few years, evolutionary genetic algorithms (eGA) were applied to solve real-life problems, and results were promising which encouraged researchers to invest more time and effort to experiment and develop and improve the used algorithms in order to emerge robust and practical optimization techniques based on GA to solve multi-objective problems.

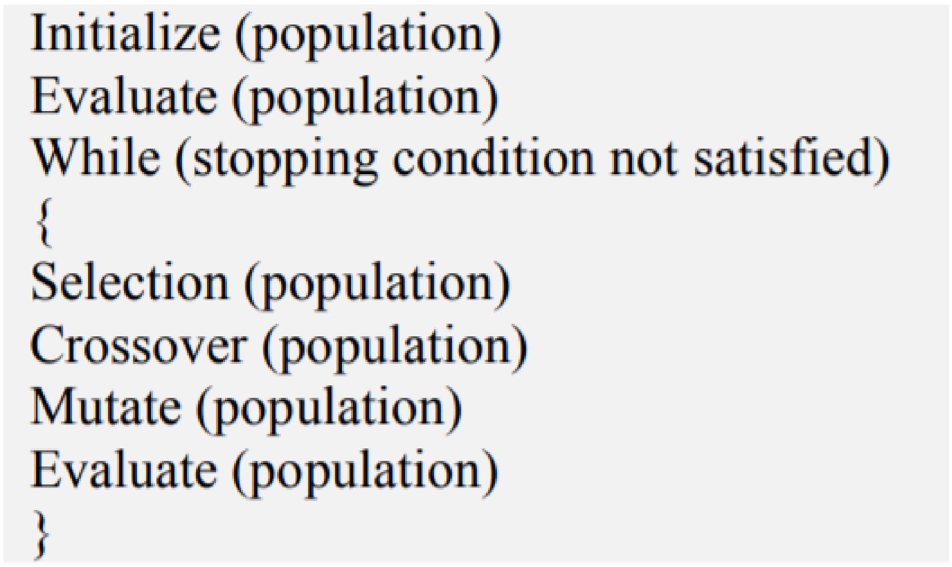
GA is classified as search algorithm inspired by nature where the formulation of new spices depends on natural selection and the fitness of individuals [43]. It also can be seen as the "abstraction of the biological evolution in computer science" [39].

GAs was first invented by John Holland back in the 1960s; where he studied the natural evolution to adopt the idea and translate it to algorithms used in computer science to solve complex problems [39]. Concepts including gene, mutation, crossover, and population are implemented in the algorithms as described in the next section.

3.2.1 eGA general Design

As explained in the previous section; the genetic algorithm inherits the natural biological concepts to formulate computer algorithms. In the living creatures; each cell is a set of chromosomes which represent the characteristics of the creature. Each chromosome has a set of genes to present a particular property or characteristic. The different possibilities for each gene are called alleles. Upon production stages; crossover, mutation, and selection are applied to generate the new offsprings. The fitness is the probability that this offspring is living and reproduce new offsprings in the next generation [39].

The computer science, the genetic algorithm is used to find the best solution from a set of solutions for the same problem which is a complex problem in most of the cases. The problem is encoded and being represented as a chromosome, a population of chromosomes is available and operators are being applied, a fitness function is well defined, in which the individuals can be evaluated and the survivors of the next generations are selected [39].



```
Initialize (population)
Evaluate (population)
While (stopping condition not satisfied)
{
Selection (population)
Crossover (population)
Mutate (population)
Evaluate (population)
}
```

FIGURE 3.5: Basic GA algorithm steps [43]

The solutions of each problem; are being represented as the population of chromosomes. The chromosome represents each one of these solutions could be a string of type binary or integer digits, each digit is called a gene. The alleles for each gene could be 0 or 1 if the gene is binary for example, where a 0 represents the absence of the gene in the given solution and the 1 means that it is present. Many representations for the chromosome can be applied; both numeric and non-numeric; but legal operators should be applied to each type of chromosomes [43] [39].

Figure 3.5 summarize the basic steps of each eGA.

3.2.2 eGA Operators

Associated with the genetic algorithms are three operators that are used in the population for the chromosomes. Operators are the like the following [43]:

1. Selection operator: Given that applying the crossover operator increases the number of chromosomes in the population; we need another operator to manage the increase of this size. The selection operator is used for determining which chromosomes to be passed to the next generation [39] based on their fitness value [43]. Chromosomes of better fitness value is passed; as they are the most likely to survive in the next generations.

2. Crossover operator: also referred to as recombination [43]. The input of the crossover process is two chromosomes which are referred to as the parents, and the output of the process is at least one new chromosome that holds some of the genes from each parent and called the offspring. Crossover is applied to random genes of the parents, and then perform changes in the selected genes either by swapping locations, or flipping bits, or reordering the sequence to generate newer offsprings. The crossover probability value is used to determine the probability and the rate to apply the crossover on a chromosome. There is a relation between the crossover probability value and the fitness value of the chromosome [39]. The higher the fitness if the chromosome means that it is selected and passed to the

next generation, then the higher the probability that the offsprings include genes from this chromosome as a parent and maybe have as well high fitness values, and it is desirable to generate offsprings with higher fitness values than their parents. It is hard to ensure this, so in most of the times, the crossover probability value is set to a higher value in correlation with the fitness value [39].

3. Mutation operator: this operator is used to introduce slight changes on the chromosomes; either by flipping the value between 0 and 1 as in the binary chromosome for specific alleles, or by swapping the location of two numbers in an integer chromosome. A constraint on the mutation process is that the resulting chromosome should be a valid one and does not violate the representation of the problem [39]. Mutation contributes to adding more diversity to the solutions as they are altering the chromosomes in the population in order to produce newer ones [43].

   The mutation process is associated with a variable named mutation probability; which referees to the likelihood of a chromosome to be mutated. The more complex when the representation if the problem is more complex; and thus sometimes correction is applied to check and repair invalid chromosomes resulting from the mutation process which means the increase of the computational time and complexity which is not desirable. To avoid this the mutation probability value is being set to be with a minimum value so the probability of performing mutation and changing chromosome is in fewer speed [39], but not low to a point we have no new chromosomes on most of the generations so finding good new solutions becomes harder.

3.3 Evaluating multi-objective algorithms and HyperVolume (HV)

Evaluating the Pareto solutions obtained when running MOEAs is an important procedure when solving complex multi-objective problems because the Pareto front has most of the time more than one solution which we need to compare them to get a solution that is satisfying to the problem objectives from the decision maker point of view. The obtained solutions are being classified based on two criteria; the closeness of the solution to the optimal Pareto front where the minimum distance between the obtained solution and the true Pareto is desired [50] and the nearer the solution is the better. The second criterion is the diversity of the obtained solution in the non-dominated front where the increase of the diversity is desired [2]. Multiple metrics were introduced in literature to evaluate the obtained solutions; each of which focuses on a specific perspective of the solution, and according to [17] no unified matrix can be used to evaluate the solution in most of the multi-objective problems. In this section, we discuss the HyperVolume (HV) matrix, as we are going to apply it for measuring the closeness to the optimal Pareto-front of the solutions obtained in this research.

3.3.1 Hyper Volume metrics

HV is one of the powerful maximization metrics used to evaluate Pareto-front solutions. The matrix gives a quantitative value that represents the difference between the size of the objective spaces obtained Pareto-front and the true Pareto-front [46]. The HV is a powerful matrix as it measures both the diversity and the closeness of the obtained Pareto-front in respect with the true Pareto-front [2]

The concept of the matrix depends on the inferior region, and hyper area value is measured by calculating the space difference between the inferior region of the true Pareto and the obtained Pareto. The objective space is occupied by the true Pareto which means a more significant inferior region than the obtained Pareto. The obtained Pareto inferior region occupies a smaller portion of the objective space. The difference between the two regions is an indication of how worse the obtained solution, and for each Pareto solution obtained the HV is calculated and the best solution is the solution with the maximum HV value [46]. Figure 3.6 represents HV equation. It states that for each solution $i \in Q$, the hypercube V_i is calculated in respect to the reference point W which is located on the true Pareto. The resulting value, i represents the diagonal corners of the hypercube.

$$HV = \text{volume}\ (\mathrm{U}_{i=1}^{|Q|} v_i)(21)$$

FIGURE 3.6: HV equation [2]

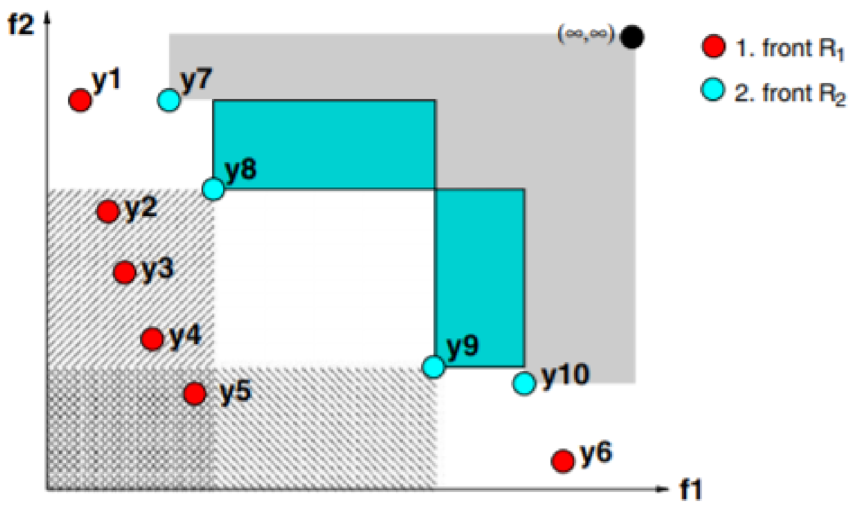


FIGURE 3.7: Visualization for the HV matrix [9]

Figure 3.7 illustrates the visualization of the HV matrix. In this figure we have two fronts; the red points represents $R_1 = y_1, ..., y_6$ and the blue points represents $R_2 = y_7, ..., y_{10}$. The light gray area represents the area dominated by R_2 where the blue rectangular represents the cubes dominated by the points y_8 and y_9 respectively; the HV value shows that the measure of y_8 outperforms the value of y_9. The hatched area shows the cubes dominated by the R_1 which are clearly contains points that dominates both y_8 and y_9 which make R_1 more interesting solution as it has more vacancy between the points y_5 and y_6 in R_1 [9].

3.4 JMetal- framework for developing metaheuristics for multi-objective optimization problems

jMetal is an open source Java-based framework introduced to facilitate researchers in solving multi-objective optimization problems; by empowering them with basic and advanced features to encode optimization problems as metaheuristics code blocks then applying algorithms and evaluating obtained solutions. The architecture of this framework is designed under object-oriented principles and a set of design goals to allow code reuse, abstraction, simplicity, portability and extensibility [18].

According to the developers of the framework; jMetal facilitates not only solving multi-objective optimization problems, but also the framework is implemented to execute different algorithms in the same pattern taking into account each algorithm properties and operators. The unified implementation using the same programming language and other development features reduce the challenges of comparing different algorithms, as numerical results are not be affected and algorithms' performance is not biased due to differences in implementation. The framework provides fair evaluations for different optimization techniques and algorithms used. This motivation helped researchers in making use of existing reliable object-oriented code to solve problems, and reducing the effect of variations of algorithms implementations on obtained results.

The main Java class under jMetal is the (Algorithm) class. All other metaheuristics algorithms inherit from this class, and each algorithm implements parameters and operators and execution methods in respect with its definition. The (Variable) class represents different representations in metaheuristics including real, binary, binary-real variables to adapt to different problems. The class (SolutionSet) is responsible for representing (Solution) obtained when executing an algorithm. The (Problem) class, represents the problem under investigation, in which the execute() method should be implemented. An important class is the (Operator) class, represents operators to be used in the evolutionary algorithms including crossover, mutation, and selection. Also, the framework provides a set of utilities like ranking population and evaluating Solution Set strength [18]. The framework is extensible, which means someone can add a costume representation for the variable, and the use of the Object) class allows high flexibility to empower the users to implement their custom classes to represent a specific problem, yet benefit from the power of the framework features.

Chapter 4

Research Methodology And Experiment Setup

To achieve planned contributions, few experiments are done on three different datasets in order to provide a solid conclusion and answers about the research questions. These experiments are implemented using jMetal [18]. This framework provides the ability to run different evolutionary algorithms which can be used to handle this research problem.

This chapter identifies the methodology used to run this study. This includes defining the objectives of the optimization of the bug fixing human resource allocation problem. Additionally, problem solutions are defined, and its structure is designed and mapped to a real planning setup.

The main factor affecting evolutionary algorithms directions and results is the solution fitness calculations. To build a Pareto optimal solution set, each solution should have a fitness value for each objective to determine the non-dominated solutions. Solution fitness calculation formulas are described in details in this chapter.

4.1 Experiment Data Sources

In order to study the SBSE evolutionary algorithms on the bug fixing human resource allocation, it is required to experiment this problem on one or more dataset to present the algorithms capabilities and answer the research questions by analyzing the results. This study provides input data for the experiment, running different algorithms on data, analyses results in order to come up with results and provide solutions through out the suggested framework.

One of the common threads to validity in many studies is the ability to validate research theory on various types of input data. Additionally, many of the researches are conducted on non-real data designed for research purposes. Additionally, many researches target open sources projects data due to the popularity and availability of data. Open source data is still a valid and significant read data source, but it does not reflect the situation, constraints, and environment of real commercial projects in the software engineering areas. A study done by Wright and others [21] has shown that almost half (49%) of recent empirical studies used solely open source projects. Such studies come up with general results and recommendations without extending the results to real projects.

In order to reduce the study validity threats and provide industry managers confidence in the study and provided framework, three different datasets are used in the experiment. Well operated companies provide two out of the three datasets. Details about the companies providing projects or any other information providing an indication about the companies or the nature of the projects they are working on are eliminated for confidentiality and business purposes.

4.1.1 Eclipse Project Data

A study done by Muhammad Rezaul Karim approached bug fixing allocation problem using Eclipse bugs dataset [27]. They have studied 2040 bugs from different 19 milestones. The data is divided into two projects:

1. ECLIPSE JDT project [19]: It provides a tool to implement the Java IDE of Eclipse to support a building of Java applications. This is the core of Eclipse Java IDE allowing it to provide a workbench of different components and tools.

2. Eclipse Platform project [20]: It provides a set of services and frameworks to support component model for Eclipse. It handles all infrastructure and resource management

Data and format:

Dataset used by Karim is available online at `https://sites.google.com/site/mrkarim/bug-data.zip?attredirects=0&d=1`. It contains Data for both Eclipse platform and JDT for 19 different milestones. Data includes in this dataset is big due to multiple milestones. In this dataset, a separate data file is used to describe the developers for each project (Platform and JDT). Table 4.1 lists the details of developers working on the JDT project.

Dev#	jdt	jface	core	swt	ui	debug	ltk	text	jdi	search	compare	other	hourlywage
1	2	2	2	1	1	0	1.5	2	0	0	4.75	2.25	90
2	1.5	3.5	1.5	2.75	2.5	0	0.75	0	0	2.75	0.25	3	100
3	1.5	2.5	2	2.25	2.25	1.5	0	0	0.75	0	0	2.25	97.5
4	1.5	2.5	2.25	2	2.5	2.25	0	0	2.25	0	0	2.25	93.75
5	2.5	0.75	2	0	0	0	0	2	0	0	0	1.5	112
6	2	2	2.25	2	1.5	0	2.75	2	0	0.75	0	2.25	100.25
7	1.5	1.5	2	1.5	1.5	2.25	0	0	3	0	0	1	87.5
8	1.5	4	1.5	2.5	2.25	0	0	2	0	0	0	1.5	75
9	1.5	3	2.25	3	2.25	0.75	0	0	0	0	0	1	93.75
10	2.25	0.5	2.25	0	0	0	1.5	2	0	0.75	0	2.5	112.5
11	1	2.25	2.25	1.5	2.25	2.75	0	0	1.5	0	0	0	81.25
12	2.25	0.5	2.75	0	0	0	0	0	0	0	0	1	125
13	2.5	0.5	1	0	0	0	0	2	0	0	0	0	87.5
14	2.5	0.5	2	0	0	0	0	2	0	0	0.5	1	100
15	1	2.5	2.25	2	2.25	2.75	0	0	2	0	0	1.5	81.25
16	2	1.5	1.5	1.5	1	0.5	2.5	2	0	0	0	2.5	87.5

TABLE 4.1: Sample of Eclipse Developers Dataset

Table 4.1 describes the details of 16 developers working on the JDT. Each developer has an id as in the first column and a wage as in the last columns. Each of the left columns describes the developer relative skill level on each of the JDT components.

Developer′s skill level is used to find the level of ability of this developer to fix a specific bug and the time required to fix it. Table 4.2 shows a sample list of bugs for the Eclipse JDT project. This table list some bugs. Each bug has a bug number. This number can be used to reference bugs

CSV files if further information is required for a bug. The second column represents the developer id which is related to developer id in table 4.1. The third column lists the effort in hours required to fix each bug for an average skilled developer. The rest of the table columns are used to describe the percentage of effort required to be put on each component for the bug to be fixed.

bug#	Dev#	Effort	jdt	jface	core	swt	ui	debug	ltk	text	jdi	search	compare	other
68552	16	46.00	0.38	0.26	0.00	0.16	0.19	0.00	0.00	0.00	0.00	0.00	0.00	0.01
69020	16	50.67	0.70	0.01	0.12	0.00	0.00	0.00	0.15	0.01	0.00	0.00	0.00	0.00
76099	17	127.33	0.26	0.28	0.09	0.09	0.14	0.00	0.00	0.00	0.00	0.14	0.00	0.00
78450	20	52.33	0.61	0.06	0.13	0.05	0.09	0.00	0.06	0.02	0.00	0.00	0.00	0.00
80784	9	8.00	0.11	0.32	0.05	0.47	0.03	0.03	0.00	0.00	0.00	0.00	0.00	0.00
92009	2	8.00	0.34	0.33	0.03	0.27	0.02	0.00	0.02	0.00	0.00	0.00	0.00	0.00
93376	20	20.33	0.89	0.03	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
101453	18	34.67	0.89	0.03	0.06	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00
101794	19	8.00	0.86	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.001

TABLE 4.2: Sample of Eclipse Bugs Dataset

These two tables are the main tables to be used for HR allocation for bug fixing plan. The main point about allocation is that a developer has a different skill level on each of the project component. The time required for a developer to fix a bug may be shorter or longer than the estimated effort based on the bug required skills.

4.1.2 Industrial Data

In order to provide more reliable results, this study targets two datasets imported from commercial companies bug tracking systems. This provides a means to target the problem from different perspectives.

In order to unify the process of running optimization algorithms on different types of datasets, it is essential to unify the data format of the input sources. The data format of Eclipse open source projects is already there and is imported from a different study while on the other hand bugs data in commercial companies are stored in bug tracking systems like Jira. In order to run the same algorithms code on different datasets, it is essential that the datasets have the same format. The more natural solution is to export data from the commercial bug tracking systems into text files with the same format as the files used for the Eclipse dataset.

Importing data from bug tracking systems and reformatting it in the required structure is a doable task, but with the selected two sources the following issues are faced:

- One of the tracking systems has exposed APIs to export bugs. Getting bugs manually is problematic due to the size of the targeted bugs and pron to manual copy faults. To handle this issue, a scraping tool is build to read the system web pages and extract the required information.
- Systems differ in the bug properties they provide. These properties are directly related to the study objectives. Missing properties leads

to missing objectives in some datasets experiments.

4.1.3 Datasets Available Bug Properties

As this study uses three different datasets. It is not necessary that all available datasets support the same bugs properties. Table 4.3 lists the supported properties by each of the study datasets.

	Commercial	Creation Date	Severity	Priority	Customer Reported
Eclipse Dataset	✗	✓	✓	✓	✗
Industrial Dataset1	✓	✓	✗	✓	✓
Industrial Dataset2	✓	✓	✓	✓	✓

TABLE 4.3: Study Datasets Properties

4.2 Experiment Setup

4.2.1 Chromosome Structure

Plan representation to be fed to different study algorithms should be readable and straightforward by evolutionary algorithms tools. To implement this, a chromosome structure is used to represent the bug assignment to developers. Additionally, It is essential to decide the order in which a certain developer fixes bugs since a developer may not be able to fix all assigned bugs in the iteration period. Consequently, a sequence number is defined and attached to the bug number to indicate the order in which the bug is fixed. Figure 4.1 illustrates a chromosome representation of a bug fixing plan.

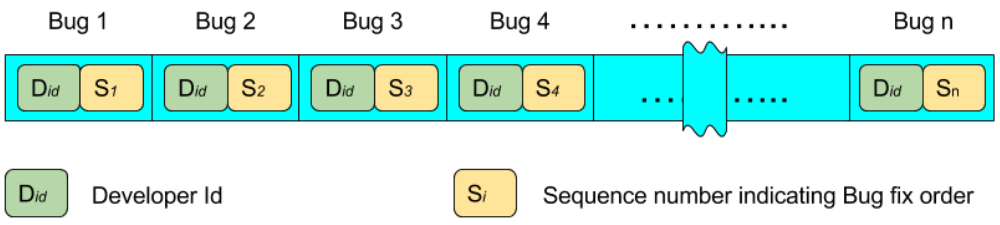


FIGURE 4.1: Experiment Chromosome Structure

The chromosome is divided into n genes representing n bugs in a backlog. The bugs gens are ordered sequentially in the chromosome. This order is always preserved withing crossover and mutation operations. Each gen in the chromosome is divided into two decimal values: Developer id and a sequence number presenting the order in which the developer fixes the bug. This number is between 0 and n-1 and should be unique in any chromosome so no two bugs have the same fix order in a solution.

4.2.1.1 Chromosome Operations

To define the operations to be valid for bug assignment chromosome, a limitation is listed on this chromosome first as the following:

1. Developer id is between 0 and the number of developers. This value may be repeated in more than one gene in the chromosome

as it is natural that the developer is assigned more that one bug to fix.

2. Number of genes is fixed and equals the total number of bugs targeted for optimization (usually the whole system bug backlog).

3. Sequence number must be between 0 and n-1 and must be presented in each gene. Each value of the sequence number is unique in the chromosome

Due to the nature and limitation of the suggested chromosome, permutation is the best choice for the chromosome variable type. Unfortunately, this chromosome is not represented by a standard permutation definition as each gene contains two values (developer and sequence) where one variable is unique, and the other one is repeatable. Consequently, a custom type of permutation should be defined. For this purpose, this study defines a mixed permutation structure and operations.

Operations for this mixed permutation chromosome is similar to the standard permutation where permutation operations conditions and limitations should be applied to the sequence value which the developer id value can be any value within the allowed range. This means that any operation should preserve the uniqueness of the sequence numbers in the chromosome.

Crossover for this mixed permutation chromosome is implemented through a two points crossover with a crossover probability. For each crossover, the two crossover points are generated randomly.

Mutation operation is a bit tricky in the mixed permutation. A swap permutation is used for this purpose. In normal swap mutation, two chromosome genes are swapped. This is used to keep the same set of genes values in a chromosome (main property of the permutation). In this study, each gene contains developer ID and sequence number. In swap operation, the sequence number is preserved to keep uniqueness where no two bugs can have the same sequence number in any solution chromosome. On the other hand, to achieve search exploration the assignment for the bug to a developer should be changed. In order to achieve this the developer id is randomly regenerated with each mutation.

4.2.2 Multi-Objective Fitness Evaluation

Every bug has an estimated time to fix ETA set by a developer or manager. Usually, this ETA is estimated based on an average skill level. A developer working on a bug that requires specific skill level on one or more competency area which she does not own or her skill level at these competency areas is low spend more time to fix it. The opposite is also valid where a developer is working on a bug requiring some competency area skill level, she is expected to fix it in a time equal or less than the bug ETA in case the developer equivalent skills levels are equal or higher than average.

To estimate the effort required by a developer to fix a bug i which has time estimation to fix ETA is a function of Bug required competency areas skill, and user skill per each component. As mathematical representation, Competency area fix time $CaFT$ for bug i is a function of effort on this Competency area c $effort(CA_c, i)$ and developed d productivity function $prod(d, i, Ca_c)$

$$CaFT(Ca_c, i, d) \quad = \quad f(effort(Ca, i) \quad , \quad prod(d, i, Ca)) \tag{4.1}$$

The time required to fix a bug i by developer d is $FT(i, d)$ is the sum of time required to fixed each of the competency areas by the developer d. Developer having low skill level on any of the competency area impacts the total time required to fix the bug

$$FT(i, d) \quad = \quad \sum_{Ca=1}^{c} CaFT(Ca_c, i, d) \tag{4.2}$$

A solution includes bugs distributed among developers trying to optimize different objectives in an iteration time period and achieving the least possible time fixing all bugs with the given resources. The bugs in the solution to be fixed by a developer should be in order. Each bug assignment is accompanied by a sequence number s_n representing the order. Equation 4.3 shows the order of bugs to be fixed by a developer

$$dev_d \quad Bugs \quad List \quad = \quad \{B_{s_1}, B_{s_2}....B_{s_n}\} \tag{4.3}$$

A developer can fix a number of bugs on the planned bugs list of the solution during iteration time T_{it}. This can be calculated by summing up the bugs fix time sequentially ($FT(i,d)$) till adding one more bug exceeds the iteration period.

$$BL_{it}(d) = \quad \{B_{s_1}, B_{s_2} \quad .... \quad B_{s_t}\} \qquad where \quad \sum_{j=1}^{t} FT(i,d)) \quad \leq \quad T_{it} \tag{4.4}$$

Based on iteration time and bugs assigned to each developer, Total bugs fixed in an iteration $B_F ixed_{it}$ is defined as a union of bugs assigned to each developer.

$$B_F ixed_{it} = \quad \cup_{d=1}^{D} \quad BL_{it}(d) \tag{4.5}$$

The bug list planned to be fixed by all the developers is used to calculate the fitness of the different objectives of this study. Competition is obvious between the first objective and the other, where high high priority may be time-consuming, hence fixing more on high priority results in a low number of total fixed bugs during the iteration.

Three out of four objectives in this study are calculated within the iteration time frame. This applies to the total number of bugs fixed in the iteration time line, the the total number of fixed high bugs in addition to the aging factor which is calculated based on the bugs fixed in the iteration. The forth objective is calculated based on the bugs that are not included in the iteration. This time can be calculated by getting the maximum of the sum of bug fix time for each developer as expressed in equation 4.7

All Bugs Assigned to developer

$$\begin{aligned} BL(d) = \quad & \{B_{s_1}, B_{s_2} \quad .... \quad B_{s_k}\} \\ \textit{where} \quad & B_{s_k}\textit{is the K bug assigned to developer} \end{aligned} \tag{4.6}$$

$$\text{All Bug Fix time TFT} \quad = max(BL(d)) \tag{4.7}$$

Objectives Values:

TABLE 4.4: Study Objectives Variables

Objective	**Optimization**
Number of fixed bugs	Max
Number of high Priority bugs	Max
Number of Aging bugs	Max
Total time to fix all bugs	Min

4.2.3 Algorithms and jMetal Study

jMetal framework [18] is used to build and run the study for both three and four objectives. The following settings are shown in table 4.5 are used for the different algorithms used in the study. Default jMetal configuration is used as tuning the algorithms for better results is not the purpose of this study.

Parameter	**Value**
Population size	100
Crossover type	Two Point
Crossover probability	0.9
Mutation type	Swap
Mutation probability	0.01
Independent runs	30
Max Evaluations	100,000

TABLE 4.5: jMetal Experiment Configuration

In this study three algorithms are used: 1) NSGA-II 2) MOCell 3) IBEA. The same parameter values are used for the three algorithms. All the runs used the same four objectives. The comparison part of this study is used to pick the best algorithm used to handle this software engineering problem.

4.2.4 Human vs. Algorithm Setup

The purpose of this experiment is to present the efficiency NSGA-II for building a bug fixing iteration plan. This is achieved by comparing the algorithm generated solutions with solutions created manually by senior level developers and managers volunteering from different companies working in the software development industry.

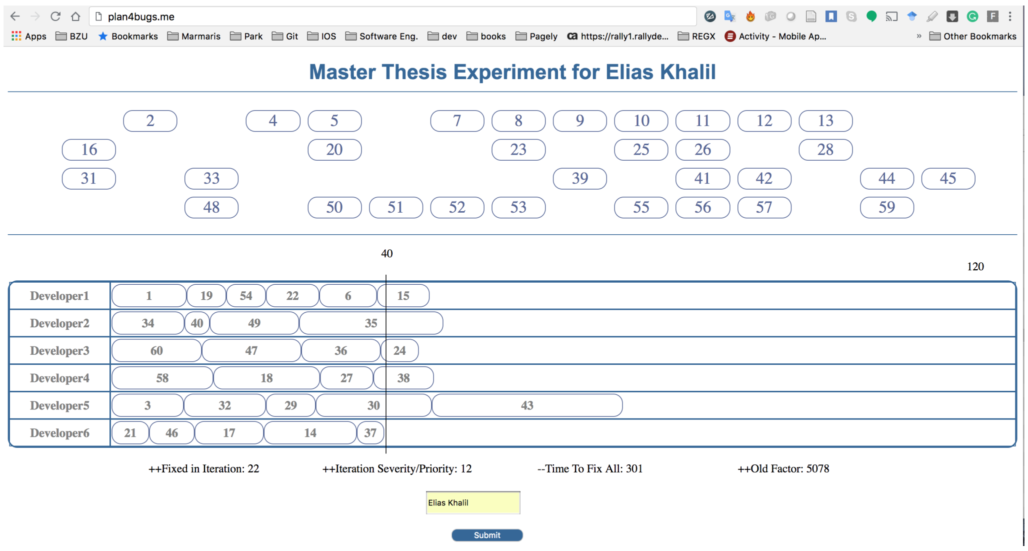
Volunteering for the experiment was requested through different Facebook and Linkedin groups involved in the software development industry. Selection of volunteers was based on the following criteria:

1. Candidate is playing management or senior development role at his career
2. Has 3+ years of related work experience
3. Involved in agile planning and estimation

The dataset of the experiment is constructed of 60 bugs with various properties and characteristics. Bugs in this dataset are collected from 5 project components: Widgets, Dashboard, Spring, Messaging and Database.

Each bug in this dataset provides required time to fix and percentage of effort required on each of the project components in addition to priority and days since the bug was created (used for aging factor objective). Another set of six developers are provided. Those developers are the expected resources to fix the bugs based on the solutions to be created. Each developer has a skill level on each of the project components. Skills are divided into levels from 1 to 5 with 5 representing the highest skill on a component.

To help volunteers in matching between bugs and developers, a special application is built for this purpose. The application is hosted on `http://plan4bugs.me`. Using this application user can see a backlog of 60 bugs. Hovering on each bug provides bugs properties. The middle part presents a table used to construct a plan. A vertical line is used as a border for the one-week iteration. For this plan, 6 developers are used to fix all bugs. Hovering on each developer provides the skill level of this developer on each of the project components.



Master Thesis Experiment for Elias Khalil

2 4 5 7 8 9 10 11 12 13
16 20 23 25 26 28
31 33 39 41 42 44 45
48 50 51 52 53 55 56 57 59

40 120

Developer1	1	19	54	22	6	15
Developer2	34	40	49	35		
Developer3	60	47	36	24		
Developer4	58	18	27	38		
Developer5	3	32	29	30	43	
Developer6	21	46	17	14	37	

++Fixed in Iteration: 22 ++Iteration Severity/Priority: 12 --Time To Fix All: 301 ++Old Factor: 5078

Elias Khalil

FIGURE 4.2: Experiment UI Tool

The experiment is about bug triage among the six developers. This is achieved by dragging the bug button to the right developer, taking into account the order of bugs to be fixed by each developer. Tooltips on bugs and developer list help the user in bug triage based on the bug required

Bug#	Effort	Widgets	Dashboard	Spring	Messaging	Database	Sevirity	Priority	Days
1	40	0.45	0	0	0	0.55	4	4	611
2	12	0.3	0.3	0.2	0.2	0	1	5	531
3	15	0	0	0.2	0.3	0.5	1	5	531
4	37	0	0.2	0.2	0.1	0.5	5	4	630
5	35	0.1	0	0.5	0	0.4	5	4	620
6	33	0.6	0	0	0	0.4	4	5	591
7	30	0.05	0.8	0.1	0	0.05	3	4	576
8	24	0.2	0	0	0	0.8	4	5	470
9	8	0	0	0	0.5	0.5	2	4	460
10	12	0	0	0.45	0.1	0.45	3	5	434
11	39	0.2	0.2	0	0	0.6	5	5	430
12	37	0.2	0.1	0.55	0	0.15	1	5	630
13	27	0.2	0.7	0.1	0	0	5	4	520
14	34	0	0.6	0	0	0.4	3	4	612
15	28	0.2	0	0	0.4	0.4	3	5	593
16	29	0.2	0.45	0.2	0	0.15	4	5	672
17	14	0.2	0.2	0.3	0.3	0	3	4	355
18	26	0	0	0.6	0.4	0	4	5	434
19	14	0.3	0	0.7	0	0	1	3	330
20	9	0.45	0	0.25	0	0.3	3	4	312
21	9	0	0	0.2	0.8	0	3	4	278
22	25	0	0	0	0.3	0.7	2	4	230
23	29	0	0	0.6	0.4	0	4	3	420
24	8	0.6	0	0	0.4	0	1	3	220
25	14	0	0.4	0.2	0	0.4	1	3	203
26	33	0.25	0.1	0.65	0	0	4	1	165
27	14	0.9	0.1	0	0	0	3	2	160
28	8	0	0	0.4	0.4	0.2	1	3	150
29	12	0	0	0	0.8	0.2	2	2	144
30	17	0.55	0	0	0	0.45	2	3	140
31	14	0	0	0	0.6	0.4	3	3	130
32	12	0.2	0	0	0	0.8	2	4	130
33	19	0.45	0	0	0.25	0.3	4	3	129
34	22	0.3	0.4	0.3	0	0	3	4	120
35	33	0	0.2	0.2	0.2	0.4	4	4	119
36	33	0	0	0.6	0	0.4	5	3	98
37	6	0.3	0	0	0.4	0.3	2	4	96
38	35	0	0	0	0	1	4	1	93
39	4	0	0.3	0	0	0.7	2	3	88
40	12	0.3	0.5	0	0	0.2	2	3	85
41	14	0	0	0.5	0	0.5	2	3	85
42	8	0.4	0.4	0	0.2	0	2	3	78
43	33	0.3	0.3	0	0.3	0.1	4	1	76
44	24	0.85	0.15	0	0	0	4	4	76
45	11	0	0	0.8	0.2	0	2	3	67
46	12	0.2	0.6	0.1	0	0.1	1	4	58
47	33	0	0.2	0	0.2	0.6	5	1	56
48	5	0	0.3	0.7	0	0	3	3	55
49	13	0	0	0	1	0	2	3	54
50	25	0.4	0	0	0	0.6	5	3	49
51	29	0	0	0	0.6	0.4	4	2	45
52	6	0.9	0.1	0	0	0	1	2	44
53	18	0	0	0.2	0.3	0.5	1	3	44
54	19	0	0	0	0.4	0.6	3	2	40
55	4	0.4	0.6	0	0	0	3	2	40
56	30	0.3	0.5	0.2	0	0	4	3	38
57	5	0.6	0.2	0.2	0	0	2	3	35
58	29	0	0.3	0.45	0	0.25	5	1	33
59	33	0.6	0	0	0	0.4	4	2	33
60	25	0.1	0.7	0.1	0	0.1	1	2	32

TABLE 4.6: Experiment Bugs Details

Dev	Widgets	Dashboard	Spring	Messaging	Database
1	5	4	2	4	3
2	3	5	1	1	2
3	1	2	4	5	2
4	2	1	3	1	4
5	1	1	2	2	1
6	1	1	1	2	2

TABLE 4.7: Experiment Developers Details

skills and developer skills. For ease of use, the application calculates the time required to fix a bug by a developer. Hence, time to fix a bug differs based on the developer fixing it. This is reflected graphically on the UI, where the width of the bug bar in the plan (similar to a Gantt chart) differs and present bug time length compared to the iteration length. The vertical line presents the iteration time border. All bugs laying on or after this border is fixed after in the following iterations and hence they are used to calculate the time to fix all bugs objective.

The lower part of the application is a live calculation of the achieved objectives. Upon each bug drag/drop, these four objectives are recalculated. These values act as guidelines for the user to optimize the plan. A ++ sign is added beside objectives to be maximized while — sign is used to indicated an objective to be minimized.

4.3 Experiment Assumptions

1. One developer per bug: This study put an assumption that each bug is assigned to one developer. This assumption is a real one as it is the way managed in industry [6].

2. competencies are set per component: Each bug requires a set of competency areas. These values should be set per bug as done in the eclipse dataset, but it is not available in the industrial data. For this study, these values are set per software module assuming that any bug in a module requires the same competency areas as the module it belongs to.

3. In this thesis Testers effort is not included in the iteration planning. It is mainly about developers plan. Testers work with developers during and after the iteration time. The time they spent of bug fixing does not affect the planning activity, so they are excluded from this study.

Chapter 5

Experiments Results and Analysis

5.1 Algorithm Comparison

The purpose of this experiment is to present the use of the HR resource allocation in the meta-heuristic domain. The experiment is indented to achieve two goals:

- Get Pareto optimal solutions for bug assignment of Eclipse JDT and platform milestones in addition to other industrial datasets. This is done by running meta-heuristic algorithms of these milestones bugs assigning them to a set of developers.
- Compare between three different algorithms to handle this problem. These algorithms are IBEA, MoCell, and NSGAII.

For the purpose of this experiment, the sets of data as shown in table 5.1 are used. The developers set is fixed on 16 developer where the number of bugs varies in each milestone bugs. Default configurations are used to run the algorithms without any tuning. All algorithms ran for 100,000 evaluation for 30 runs.

	Number of Bugs	**Number of Developers**
JDTMilestoneM2	82	16
JDTMilestoneM3	142	16
JDTMilestoneM4	244	16
JDTMilestoneM5	259	16
Industrial Dataset1	235	16
Industrial Dataset2	243	16

TABLE 5.1: Experiment Datasets Details

The experiment was run using jMetal4.5. The chromosome designed in section 4.2.1 is implemented using a custom permutation called Mixed Permutation. Both two-point permutation crossover and swap mutation are extended to support a permutation containing both developer id and bug order.

To measure the quality of the produced Pareto front of each algorithm, HV quality indicator is used for this purpose. HV is calculated for each run on each algorithm. Hence, for 30 executed runs, each algorithm has 30 value for HV. jMetal experiment is used to collect these values and calculate the HV for each algorithm. This is repeated for each targeted milestones bugs. Table 5.2 displays the mean HV values.

	NSGAII	**IBEA**	**MoCell**
JDTMilestoneM2	$4.91e-01$	$8.03e-02$	$4.81e-01$
JDTMilestoneM3	$8.68e-01$	$5.98e-01$	$8.55e-01$
JDTMilestoneM4	$8.09e-01$	$5.85e-01$	$7.69e-01$
JDTMilestoneM5	$6.30e-01$	$4.91e-01$	$7.04e-01$
Industrial Dataset1	$4.49e-01$	$4.42e-01$	$4.46e-01$
Industrial Dataset2	$2.31e-01$	$1.95e-01$	$7.04e-01$

TABLE 5.2: HV. Mean values

Table 5.2 shows that IBEA always has the least HV in all experiments. The difference between the IBEA and NSGAII and MoCell is always significant. While both NSGAII and MoCell have close values. In milestone 2 to NSGAII slightly outperforms MoCell while it is the opposite in milestone 5. Consequently, it is obvious that both NSGAII and MoCell are having close performance on all provided datasets. The difference in the results does not make any preference for any algorithms. Further experimentation is required on this aspect with different sizes of datasets and different numbers of developers to come up with a more solid conclusion.

For a better graphical representations of the results, plot-box charts are used to compare HV range and mean for each dataset. This is illustrated in Figure 5.1, Figure 5.7, Figure 5.8, Figure 5.9and Figure 5.9.

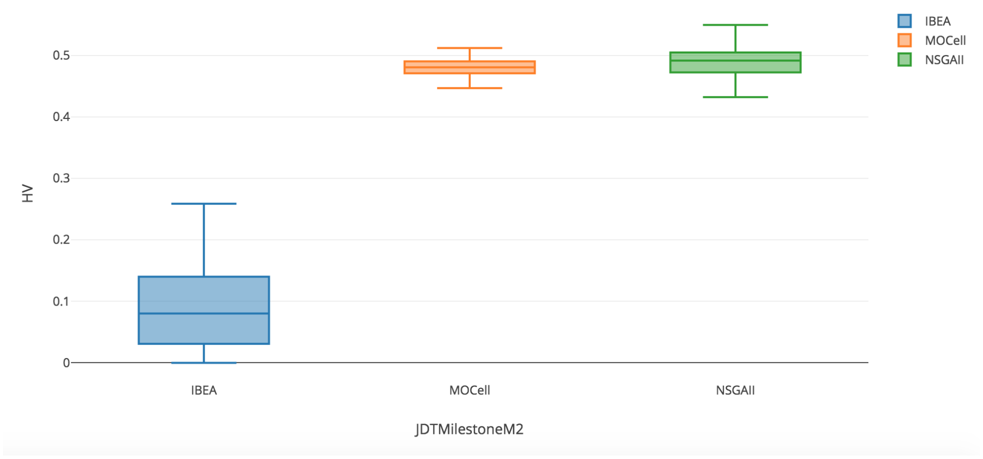


FIGURE 5.1: HV Quality Indicators for JDTMilestoneM2

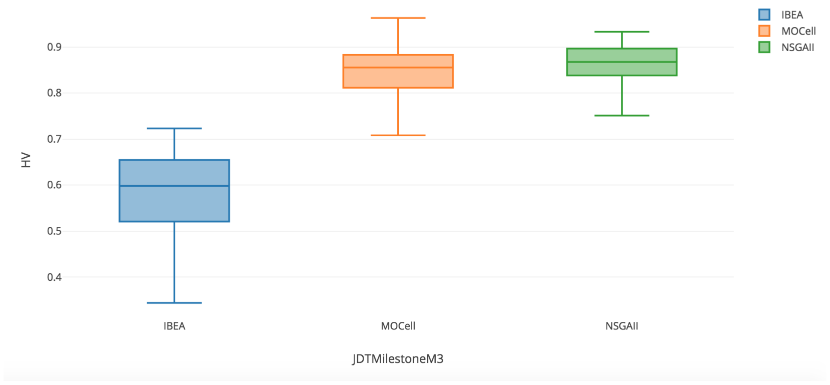


FIGURE 5.2: HV Quality Indicators for JDTMilestoneM3

Industrial data showed fewer differences between the different algorithm use for handling planning problem. These differences are not significant. Additionally, industrial data comply with open source data in regards to the fact the NSGAII slightly outperforms the other algorithm for this problem. No differences are found between open source and industrial data conclusion, but this including industrial data added more credibility to the results as having different sources for data makes the

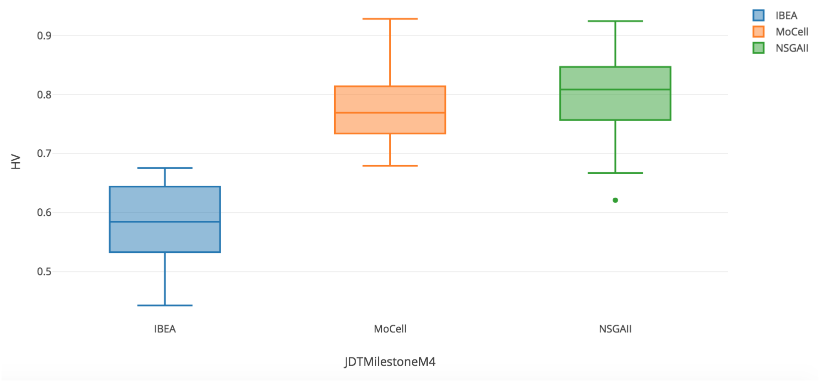


FIGURE 5.3: HV Quality Indicators for JDTMilestoneM4

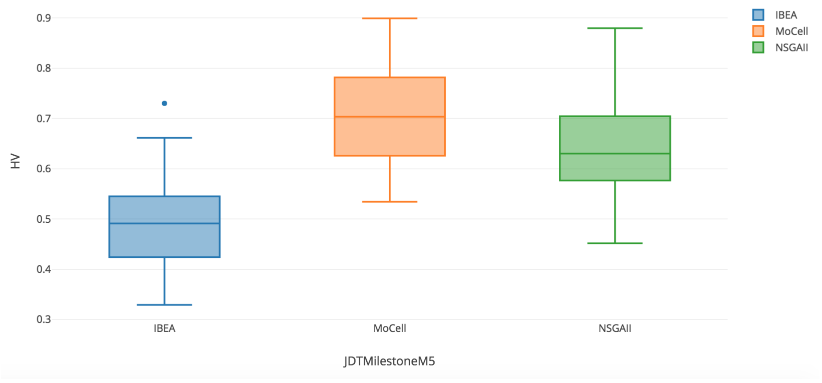


FIGURE 5.4: HV Quality Indicators for JDTMilestoneM5

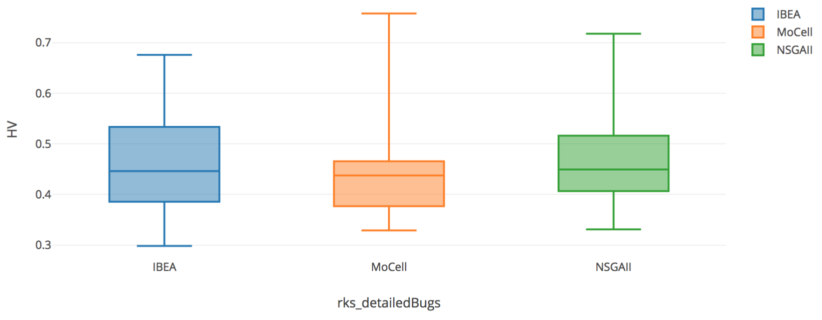


FIGURE 5.5: HV Quality Indicators for Industrial Dataset 1

experiment more realistic and more generalized.

Based on the above results HV measurements for each of the datasets

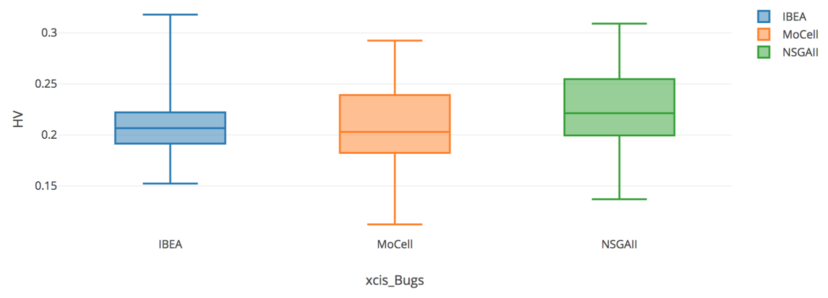


FIGURE 5.6: HV Quality Indicators for Industrial Dataset 2

on each algorithm, these algorithms can be ranked. The Ranking of algorithms is calculated based on Friedman statistic considering reduction performance (distributed according to chi-square with 2 degrees of freedom: 6.5). NSGA-II has the highest ranking of 2.75 followed by MoCell with a ranking of 2.25 while IBEA. Based o these results, the rest of the experiment is run based on NSGA-II which achieves a higher score, and hence it is considered as the best choice for optimization of bug resource allocation problem.

Algorithm	**Ranking**
NSGAII	2.75
MOCell	2.25
IBEA	1.0

TABLE 5.3: Average Rankings of the algorithms

5.1.1 Algorithm Minimum Run Time

Based on section 5.1, it is obvious that NSGA-II is the best algorithm out of the three experimented algorithms. The second goal is to determine the minimum time to get near-optimal solutions for NSGA-II optimization. To do so, NSGA-II can be run on the datasets with predefined stopping time. This approach is used to collect data for the algorithm running for incremental periods.

To determine the minimum time required to run the NSGA-II algorithm on a dataset and get a close to optional Pareto of solutions. The optimization algorithm is run over three datasets (JDTMilestoneM2, JDTMilestoneM4, and JDTMilestoneM5). The run is paused at intervals between zero and 300 sec and measured HV on each pause. It is repeated for 5 runs on each dataset. Then estimated the mean HV values at each time interval. HV mean values over time for the five datasets. HV values over time are collected in Table 5.4. This table lists the median values for 5 runs on each dataset. The results is illustrated in figure 5.7, figure 5.8, figure 5.9 , figure 5.10 and figure 5.11.

Based on the HV over time results for the three datasets, it is evident that HV increases by time till it reaches a steady state where additional time running the algorithm does not add significance to the HV values. The time when the HV converge to a steady state value differs based on the dataset time. For this study and the following experiment, the 150 sec is considered as the convergence point as at 150 sec the HV value is very close to the optimal value.

Time	MilestoneM2	MilestoneM4	MilestoneM5	Dataset1	Dataset2
5	0.302	0	0	0.26	0
10	0.376	0.013	0	0.301	0
15	0.406	0.029	0.004	0.341	0.01
20	0.419	0.05	0.016	0.33	0.021
25	0.432	0.079	0.031	0.376	0.046
30	0.434	0.099	0.055	0.401	0.072
35	0.437	0.137	0.097	0.446	0.094
40	0.452	0.158	0.132	0.436	0.107
45	0.457	0.205	0.157	0.44	0.112
50	0.453	0.229	0.165	0.427	0.117
55	0.458	0.274	0.185	0.442	0.124
60	0.463	0.3	0.216	0.43	0.132
65	0.462	0.329	0.247	0.425	0.144
70	0.479	0.365	0.271	0.425	0.149
80	0.481	0.422	0.323	0.433	0.158
90	0.481	0.479	0.378	0.428	0.162
100	0.486	0.558	0.419	0.431	0.19
120	0.486	0.639	0.492	0.431	0.235
150	0.486	0.769	0.558	0.431	0.236
180	0.486	0.793	0.598	0.431	0.235
210	0.486	0.793	0.598	0.431	0.244
240	0.486	0.793	0.598	0.431	0.241
300	0.486	0.793	0.598	0.431	0.243

TABLE 5.4: HV convergence overtime for diffirent datasets

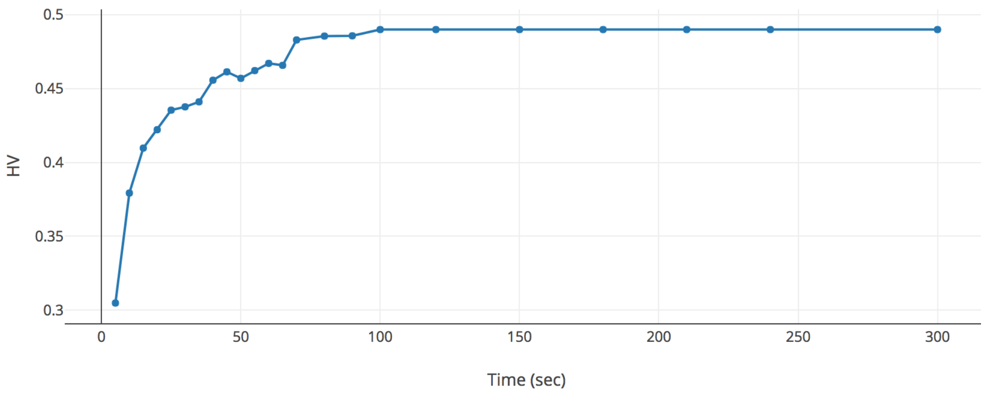


FIGURE 5.7: Run Time vs HV for JDTMilestoneM2

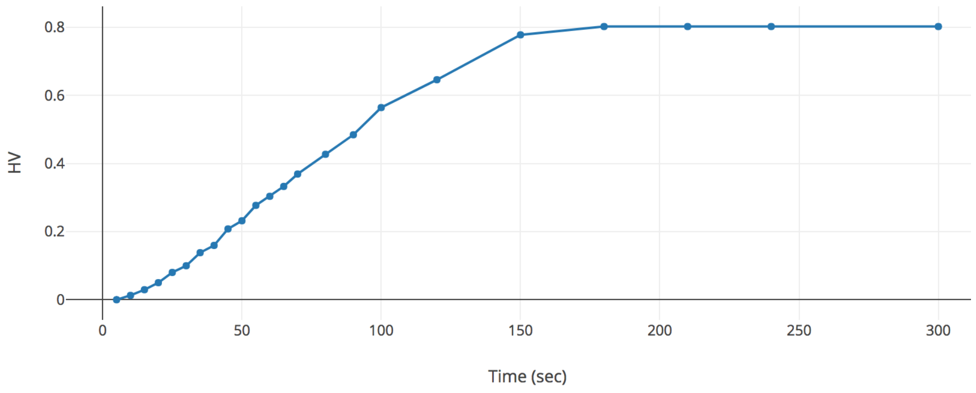


FIGURE 5.8: Run Time vs HV for JDTMilestoneM4

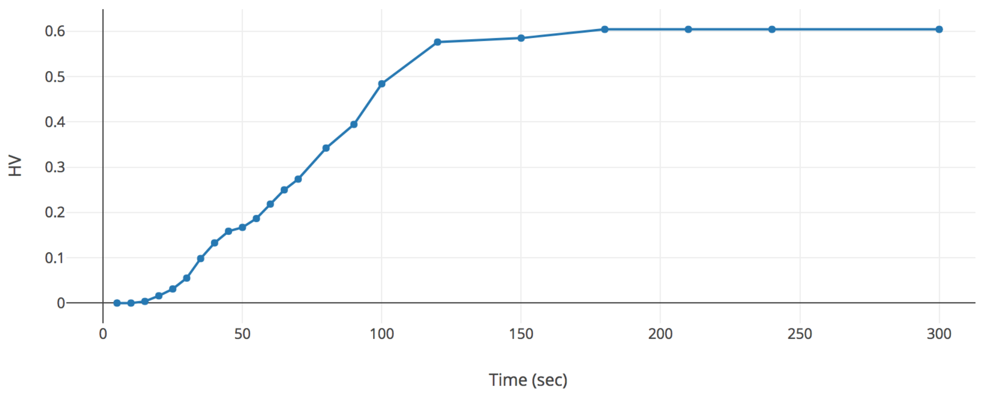


FIGURE 5.9: Run Time vs HV for JDTMilestoneM5

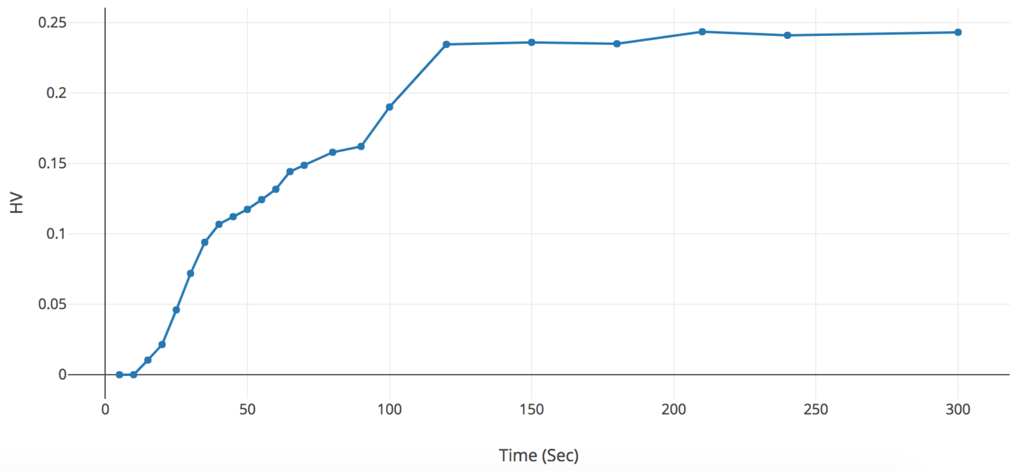


FIGURE 5.10: Run Time vs HV for Industrial Dataset1

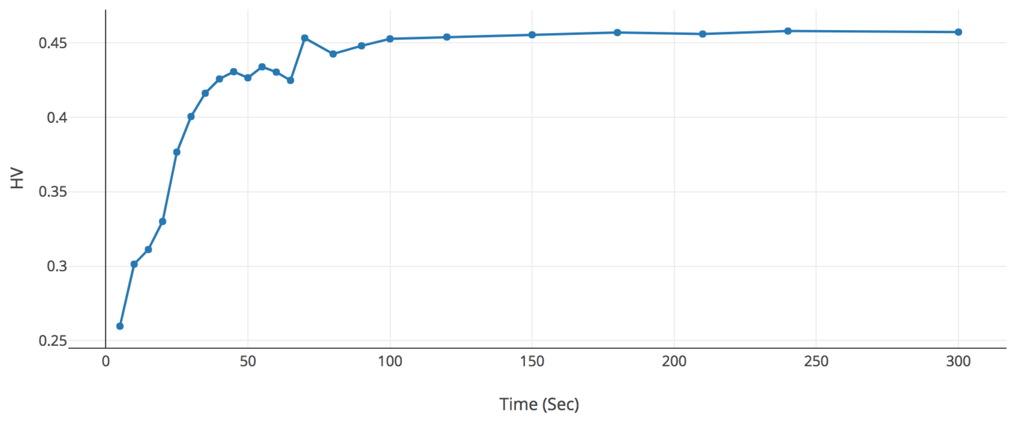


FIGURE 5.11: Run Time vs HV for Industrial dataset2

5.1.2 Human vs. SBSE Experiment

In this part, 31 volunteers participated in building plans for assigning 60 bugs to 6 developers in 1-week iteration period. This was done in 2 lab sessions for 18 volunteers, in addition to 13 online volunteers. Orientation was given to participants, and they were asked to build a plan using the application. Each user was allowed to submit up to 5 solutions. The setup was explained in section 4.2.4.

The time it took each user to build and submit 5 different plans for these 60 bugs using the application was between 45 to 67 minutes for 61% (19 users), while 26% (8 users) were able to complete it between 30 to 45 minutes. These time values are considered quite long taking into account the size of the backlog (60 bugs) and the ease of the experiment application. In comparison, building a set of non-dominated solutions using NSGA-II only takes 150 seconds.

Solutions collected from experiment volunteers were 146 solutions.

All these solutions are gathered in one dataset and removed out all dominated solutions to end up with 38 non-dominated solutions. These filtered solutions represent a Pareto front for human-made solutions. This human-made Pareto front is compared against NSGA-II Pareto front.

NSGA-II was run 30 times on the 60-bugs dataset. Each run took 150 sec to complete. This has produced 30 typical near-optimal Pareto fronts of 100 solutions each. For comparison with the human-made Pareto, one NSGA-II Pareto front with the closest HV value to the mean of the 30 HV values is selected. Table 5 shows the HV quality indicator for the Human-made Pareto in addition to the mean value of HV of the 30 runs of NSGA-II. This shows in a definite way how NSGA-II is superior to the human in building a bug fixing iteration optimizing the 4 study objectives.

Result Source	**HV (Mean Value)**
Human-made	0.358344
NSGA-II runs	0.427252

TABLE 5.5: Hypervolume for NSGA-II vs Human-made Paretos

As the optimization problem in this study is a 4-objective problem, it is not possible to compare 4d Pareto graphs. To illustrate the difference between the two approaches and show how NSGA-II outperforms the human approach, the 4d Pareto is projected into 3 different illustrations. Each figure shows the first objective (total bugs covered in the iteration) in the x-axis, while another objective is drawn on the y-axis. Those relationships are shown in Figure 5.12, Figure 5.13 and Figure 5.17 respectively. In these charts, blue circle dots represent the NSGA-II results while the yellow star dots represent the human-generated results.

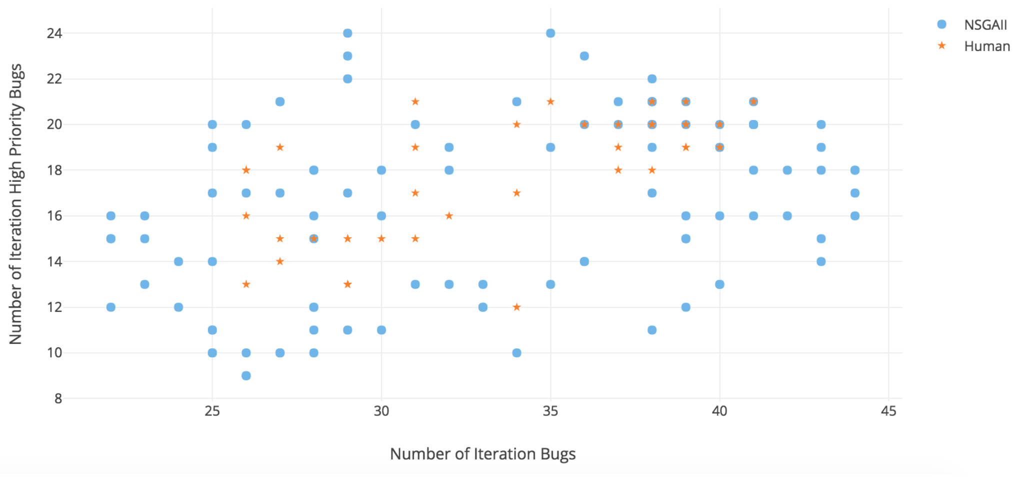


FIGURE 5.12: Total Bugs Vs Priority 2d Projection

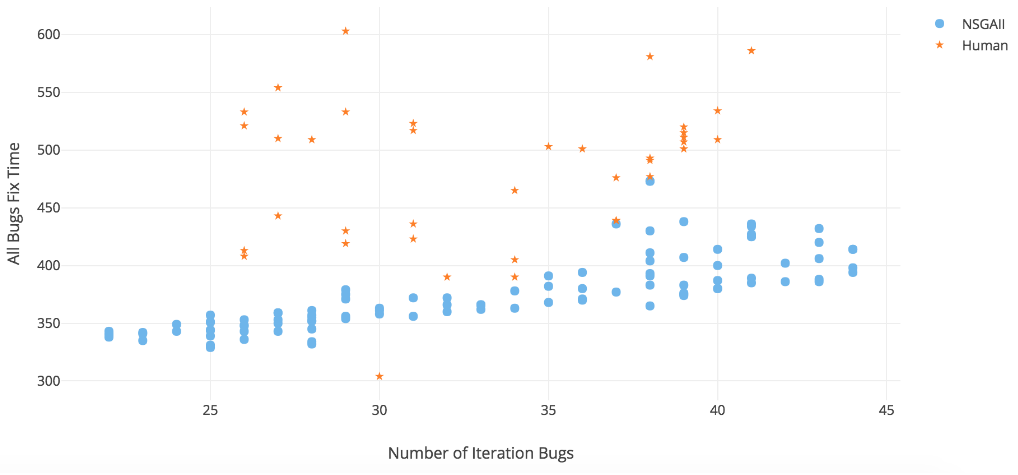


FIGURE 5.13: Total Bugs Vs Time 2d Projection

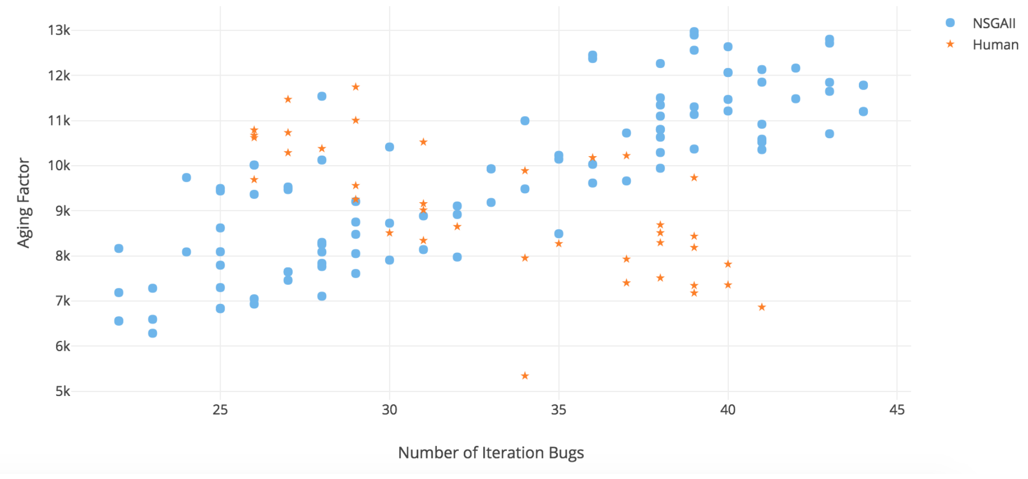


FIGURE 5.14: Total Bugs Vs Aging 2d Projection

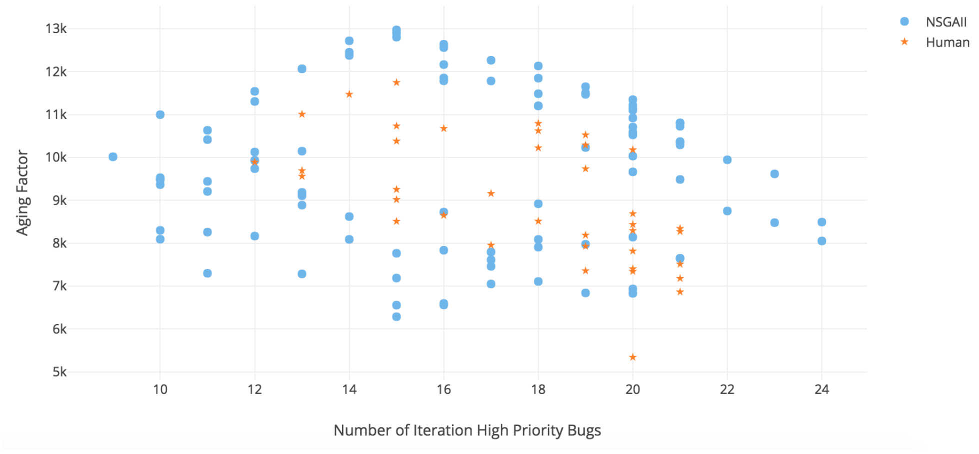


FIGURE 5.15: Priority Bugs Vs Aging Factor 2d Projection

Figure 5.13 and Figure 5.17 clearly show the superior results of NSGA-II. In human based planning, volunteers were hardly looking into long-term objectives such as time. The same applies to the aging factor where NSGA-II can handle aging factor much better as we can notice the higher NSGA-II values while human results for aging factor are significantly lower.

On the other hand, Figure 5.12 shows that human managers achieved

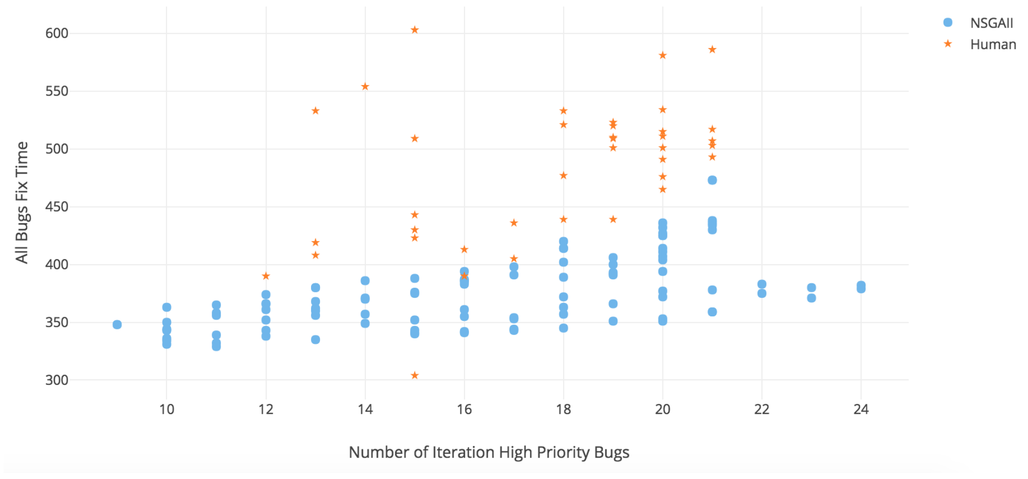


FIGURE 5.16: Priority Bugs Vs Time left to Fix all bugs 2d Projection

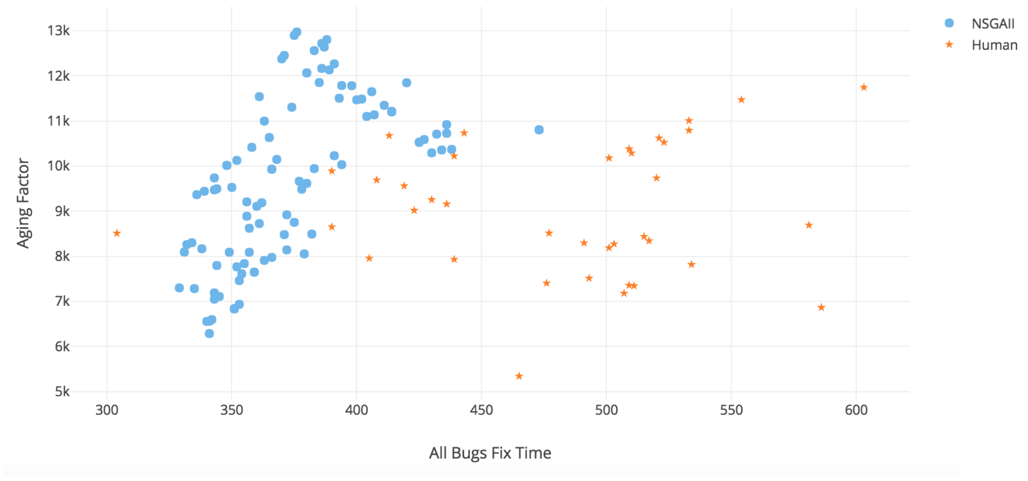


FIGURE 5.17: Time left to Fix all bugs Vs Aging Factor 2d Projection

results competitive to NSGA-II even though NSGA-II results are still higher. This shows that iteration planners tend to look at the total number and priority of bugs while paying less attention to other objectives, even though the orientation covered all 4 objectives before the experiment.

5.2 Framework Planner Advisor UI Tools

5.2.1 Solutions Radar Charts

Based on the above results, it is obvious that our bug iteration planner meta-heuristics approach has a significant help in providing decision makers with a set of solution they can select from in order to come up with a plan that achieves the best optimal values on the four objectives. The results are displayed in a jMetal FUN file. The FUN file represents the objective values for each solution in the Pareto Optimal generated by running the meta-heuristics algorithm [**a**]ltnebro2013jmetal. Table 5.6 represents a sample for the fun file.

Reading the FUN file is not trivial and selecting solutions out of the jMetal output is not easy and wont be practical. Many way can be used to simplify the process of selecting few solutions. A good solution to handle the complexity of reading the Pareto results and comparing objectives is Radar charts. At this stage, with each run a radar chart is generated by the framework, selecting 4 different solutions from the FUN to ease the selection process on the decision maker. This chart contains an interactive legend where the user can add or remove a solution by clicking on its legend. A sample of such interactive radar chart can be shown on `http://plan4bugs.me/radar.html`

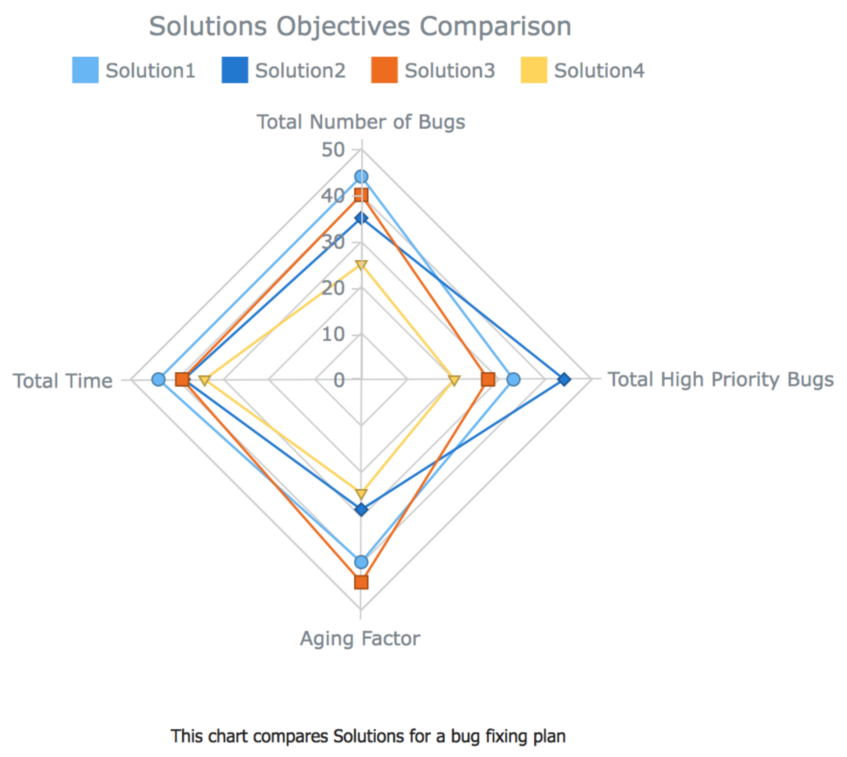


FIGURE 5.18: Solutions Radar Chart

Bugs	Priority	Aging Factor	Time
-25.0	-11.0	-7299.0	329.0
-34.0	-24.0	-8892.0	381.0
-38.0	-14.0	-13015.0	427.0
-44.0	-17.0	-11827.0	441.0
-36.0	-18.0	-12109.0	466.0
-25.0	-14.0	-6361.0	331.0
-19.0	-10.0	-10043.0	340.0
-26.0	-15.0	-6881.0	332.0
-29.0	-21.0	-7118.0	358.0
-43.0	-16.0	-12336.0	424.0
-37.0	-23.0	-9342.0	386.0
-40.0	-12.0	-11397.0	368.0
-21.0	-12.0	-9542.0	339.0
-28.0	-19.0	-8034.0	351.0
-31.0	-23.0	-7989.0	370.0
-32.0	-24.0	-7871.0	379.0
-41.0	-21.0	-9514.0	380.0
-34.0	-19.0	-11072.0	451.0
-24.0	-17.0	-9147.0	365.0
-35.0	-18.0	-11979.0	460.0
-20.0	-11.0	-9744.0	340.0
-22.0	-13.0	-9451.0	349.0
-42.0	-20.0	-10047.0	379.0
-36.0	-10.0	-11266.0	366.0

TABLE 5.6: Partial Sample of a jMetal FUN File

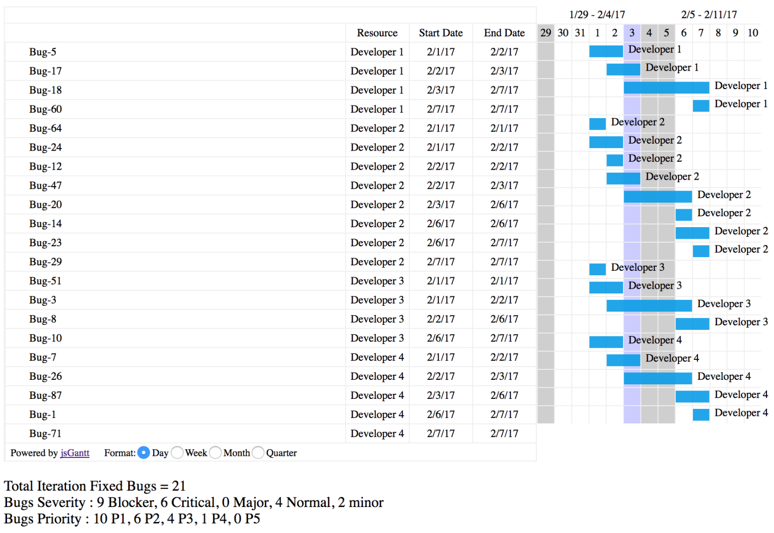
5.2.2 Gannt Plan Chart

As the framework is using jMetal, results for solutions in jMetal are stored in a VAR file as shown in figure 5.19. This file is hard to read even for jMetal experts. Each field of the solution represents developer, order number and bug number consequently. This raises the need to build a UI tool to present a solution (plan).

```
{0,1,18}  {0,2,37}  {2,3,39}  {0,4,30}  {3,5,32}  {0,6,4}   {0,7,15}  {3,8,60}  {2,9,7}
{2,10,40} {0,11,44} {0,12,26} {0,13,51} {0,14,49} {0,15,52} {0,16,16} {0,17,25} {2,18,3}
{3,19,19} {0,20,35} {2,21,20} {0,22,10} {2,23,11} {0,24,34} {0,25,41} {0,26,13} {0,27,21}
{2,28,8}  {2,29,42} {0,30,24} {0,31,14} {3,32,56} {0,33,29} {0,34,9}  {0,35,22} {3,36,1}
{0,37,28} {3,38,23} {5,39,38} {1,40,5}  {3,41,50} {0,42,17} {0,43,47} {0,44,12} {2,45,27}
{0,46,54} {0,47,31} {2,48,55} {2,49,53} {0,50,46} {0,51,2} {1,52,59}  {2,53,57} {0,54,0}
{0,55,36} {0,56,58} {0,57,6} {0,58,43} {0,59,33} {0,60,48}
```

FIGURE 5.19: Record

This framework provides a utility to convert jMetal solution into an HTML Gannt chart for this solution. This helps the plan decision maker to visually as shown in figure 5.20



	Resource	Start Date	End Date
Bug-5	Developer 1	2/1/17	2/2/17
Bug-17	Developer 1	2/2/17	2/3/17
Bug-18	Developer 1	2/3/17	2/7/17
Bug-60	Developer 1	2/7/17	2/7/17
Bug-64	Developer 2	2/1/17	2/1/17
Bug-24	Developer 2	2/1/17	2/2/17
Bug-12	Developer 2	2/2/17	2/2/17
Bug-47	Developer 2	2/2/17	2/3/17
Bug-20	Developer 2	2/3/17	2/6/17
Bug-14	Developer 2	2/6/17	2/6/17
Bug-23	Developer 2	2/6/17	2/7/17
Bug-29	Developer 2	2/7/17	2/7/17
Bug-51	Developer 3	2/1/17	2/1/17
Bug-3	Developer 3	2/1/17	2/2/17
Bug-8	Developer 3	2/2/17	2/6/17
Bug-10	Developer 3	2/6/17	2/7/17
Bug-7	Developer 4	2/1/17	2/2/17
Bug-26	Developer 4	2/2/17	2/3/17
Bug-87	Developer 4	2/3/17	2/6/17
Bug-1	Developer 4	2/6/17	2/7/17
Bug-71	Developer 4	2/7/17	2/7/17

Powered by jsGantt Format: [✓ Day] Week Month Quarter

Total Iteration Fixed Bugs = 21
Bugs Severity : 9 Blocker, 6 Critical, 0 Major, 4 Normal, 2 minor
Bugs Priority : 10 P1, 6 P2, 4 P3, 1 P4, 0 P5

FIGURE 5.20: A Solution Gannt Chart

5.3 Results Impact and Use

This study provide a detailed analysis for using meta-heuristic algorithms to solve resource allocation for a bug fixing iteration. A framework is suggested to handle solving the optimization problem of the planning. Throughout this study, experiments are conducted to present the advantages of using this framework. These advantages have direct impact on industrial environment.

Putting this framework in industrial use does not require significant efforts but it requires some changes on the way bugs are handled. The following list describes the requirements to put the framework in use:

1. Bugs backlog should provide the following proprieties in order to be able to extract and build the framework algorithm chromosome:

   (a) Bug Creation time: to handle the aging fitness factor

   (b) Bug priority and/or severity required to calculated the number of high priority/severity bugs in the target iteration plan

   (c) Estimated time to fix the bug

   (d) Product components: used to divide bug required effort per component

   (e) Percentage of effort per component required to fix the bug. This is the major information which is missing in most bug repositories while it is essential for the framework. AI may be put in use to provide effort percentage per component but still in the worst case it can be provided manually per bug

2. For each developer, skill set should be provided based on the product components.
3. Bug repository and management tool should support APIs to extract bugs and developers details. Additionally more APIs should be provided to set bug assignment and building an iteration plan

The main effort is to put extract the data from the bug repository in the format provided in table 4.1 and table 4.2. Providing such data is sufficient in order to run the framework. The minimum time required for run is 150 sec but this period may be extended in case of repositories of sizes larger than the data sets targeted in this study. This study provides time analysis as illustrated in figures 5.7 to figure 5.11. These charts shows that the 150 sec time is enough to get a stable solution for different data sets sizes.

The experiment that is conducted to compare human made plan with the framework generated plans shows the need for such framework where the framework was able to achieve better results than manual planning. Based of the analysis of the results, it is clear that human does not pay attention to all objectives of the plan. They do focus on the straightforward objectives such as number of bugs achieved. This framework makes a balance and make it easy for the managers to select solutions with optimal values. For example, the best value of aging factor achieved by human was 11.8k while the framework was able to achieve 13k aging factor fitness. The same applies for time required to fix all bugs where based on human plans the minimum time to fix all bugs was more than 600 hours while it was around 470 hours using the framework. This shows the importance of the framework to achieve better results

Chapter 6

Conclusion And Future Work

6.1 Threats to Validity

In this section, we discuss possible threats to construct, external, and conclusion validity.

Construct validity threats are concerned with the chosen objectives and the way they are calculated. We relied on measurements reported in public datasets that were accepted and used by the research community. We defined our objectives differently because we adapted them according to the agile iteration process, while previous studies only considered fixed budget and resources.

External validity threats are related to the applicability and generalizability of the results. While this study is confined to the bug-developer assignment in the agile iteration scenario, it has many similarities to other problems in search-based software engineering (SBSE), where many objectives are competing against each other, and managers are faced with large and complex decision spaces that render manual assignments impractical and inefficient.

Threats to conclusion validity can arise from the intrinsic randomness within the meta-heuristic algorithms, as well as the variations in human performance when placed under experiment. On the algorithm side, we made sure to run the algorithms 30 times on each instance of the problem and utilized the Friedman test to rank the algorithms statistically. On the human side, we obtained bug iteration plans from 31 experienced managers and senior developers, each providing 5 different attempts, and those solutions were then filtered down to 38 non-dominated solutions that we compared with the automated results. This is how we ensured the soundness of conclusions.

6.2 Conclusion

The importance of this study is presented through providing an automation framework for a bug fixing iteration planning which can provide managers and planners with a Pareto front of solutions to select from based on their preferences of objectives while making sure any selected solution is optimal and non-dominated. To get the best optimization for this software engineering problem, we have tested the problem on three different well known meta-heuristic algorithms over various dataset sizes. NSGA-II outperformed both IBEA and MoCell. Additionally, Minimum time to produce a near to optimal Pareto was extracted by running the optimization over time and measuring the hypervolume quality indicator over time. Planning bug fixing iteration through SBSE is a way more effective than manual planning. This is proven through an experiment of comparing solutions built by NSGA-II running for 150 seconds with solutions made by senior developers and managers with experience with agile planning. Both NSGA-II and human run this experiment on the same dataset showed that all human-made solutions were less optimal than the SBSE approach. Additionally, it showed that humans tend to concentrate on one or two straightforward and essential objectives while paying less attention to other objectives which results in less optimal solutions.

6.3 Difficulties and Obstacles

The main difficulty faced in this study is the ability to import data from industrial bug repositories. This is due to the fact that these systems do not provide the ability to import data through proper APIs. To overcome this issue, Repositories Web pages are used to scrape the right data and format it in a proper way to be used by the thesis algorithms.

Another difficulty faced during the thesis is the amount of computation power required to run experimentation. This research depends on different datasets with different data sources. Additionally, algorithms have to be tuned and upon any bug, fix experimentation should be repeated. Also to mention that each experiment was executed for at least

30 runs. To handle such need for computational power, AWS cloud was used to run the work in parallel where many instances of the experiments were run concurrently and data were collected afterward.

6.4 Future Work

6.4.1 Man in the loop support

In all experiments run in this thesis, the results were always represented as a Pareto front of non-dominated solutions. Based on jMetal settings, the population size was 100 solution and based on the high possible solutions, a Pareto front was always a set of 100 non-dominated solutions. These solutions represent the best solutions from where a manager or planner can select solutions based on their preferences while preserving the fact of being near to optimal.

In order to reduce the choices for the plan selector, the framework provides the users with five solutions. Four solutions are picked from the edges of the four objectives while the fifth solution is selected around the mean values of all objectives. The Framework provides these solutions as radar chart which is suitable for presenting the 4 objectives on a 2d chart as shown in figure 5.18. Another helpful output was a Gantt chart for a selected solution as illustrated in figure 5.20. This gives the manager a GUI sight to the bug assignment among developers.

Both Gantt and radar charts are so helpful outputs of the framework, but they keep the manager with few solutions to select for. Moreover, a manager may not be satisfied with the provided solutions and prefers other solutions fulfilling some criteria or objectives. This can be achieved by giving the manager the ability to influence the algorithm flow while it is running. This approach is called "man in the loop" approach.

Man in the loop could be very helpful in bug fixing planning. This can be achieved by pausing the algorithm execution each n seconds to allow the user to set some preferences. One good preference is fixing a bug assignment to a specific developer. This should not be harmful as assigning a bug to this developer is already one of out optimized solutions. The algorithm execution is resumed after one or bug assignments is fixed. After few iterations, most bugs assignments will be fixed with limited number of Pareto solutions. This approach is supposed to results in one or few solutions which are mostly human-directed while at the same time are near optimal.

In order to investigate human in the loop approach, it is required to compare the quality of solutions that result from normal optimization without any human intervention with main in the loop approach. If the normal Pareto does not dominate the man in the loop results and it has equal or better quality indicator, it can be considered a significant contribution to the optimization problem.

6.4.2 Integration with Management tools

This research proves the significant help that SBSE can provide for managers or planners to build a bug fixing iteration. This thesis has proved this by using different datasets from open source and industrial projects. This contribution raises the need to integrate this framework with modern management and agile tool in order to automate the whole process.

The integration between management tools and my framework can be divided into three parts:

1. Data expert from management tool. This can be achieved by using tool API's to get bugs and developers information and build the main files for bugs and developers that are compatible with the framework.

2. Running the framework and getting a Pareto front of a solution. Provide the users with a UI interface to pick a solution of the Pareto.

3. Import the selected solution into the management tools. This can be achieved using the management tools provided APIs.

Bibliography

[1] Jesús S Aguilar-Ruiz et al. "An evolutionary approach to estimating software development projects". In: *Information and Software Technology* 43.14 (2001), pp. 875–882.

[2] Abdel Rahman Ali M Ahmed, Mohamed H Gadallah, and HeshamA Hegazi. "Multi-Objective Optimization Indices: A Comparative Analysis". In: *Australian Journal of Basic and Applied Sciences* 8.4 (2016), pp. 1–12.

[3] Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. "Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine". In: *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on*. IEEE. 2009, pp. 216–221.

[4] John Anvik. "Automating bug report assignment". In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 937–940.

[5] John Anvik, Lyndon Hiew, and Gail C Murphy. "Who should fix this bug?" In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 361–370.

[6] John Anvik, Lyndon Hiew, and Gail C Murphy. “Who should fix this bug?” In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 361–370.

[7] Qinghai Bai. “Analysis of particle swarm optimization algorithm”. In: *Computer and information science* 3.1 (2010), p. 180.

[8] Victor Basili et al. “Understanding and predicting the process of software maintenance release”. In: *Proceedings of the 18th international conference on Software engineering*. IEEE Computer Society. 1996, pp. 464–474.

[9] Nicola Beume, Boris Naujoks, and Michael Emmerich. “SMS-EMOA: Multiobjective selection based on dominated hypervolume”. In: *European Journal of Operational Research* 181.3 (2007), pp. 1653–1669.

[10] Pamela Bhattacharya, Iulian Neamtiu, and Christian R Shelton. “Automated, highly-accurate, bug assignment using machine learning and tossing graphs”. In: *Journal of Systems and Software* 85.10 (2012), pp. 2275–2292.

[11] Nazia Bibi, Ali Ahsan, and Zeeshan Anwar. “Project resource allocation optimization using search based software engineering—A framework”. In: *Digital Information Management (ICDIM), 2014 Ninth International Conference on*. IEEE. 2014, pp. 226–229.

[12] Nazia Bibi, Zeeshan Anwar, and Ali Ahsan. “Comparison of Search-Based Software Engineering Algorithms for Resource Allocation

Optimization". In: *Journal of Intelligent Systems* 25.4 (2016), pp. 629–642.

[13] Barry Boehm, Chris Abts, and Sunita Chulani. "Software development cost estimation approaches—A survey". In: *Annals of software engineering* 10.1-4 (2000), pp. 177–205.

[14] Yguaratã Cerqueira Cavalcanti et al. "The bug report duplication problem: an exploratory study". In: *Software Quality Journal* 21.1 (2013), pp. 39–66.

[15] Wei-Neng Chen and Jun Zhang. "Ant colony optimization for software project scheduling and staffing with an event-based scheduler". In: *IEEE Transactions on Software Engineering* 39.1 (2013), pp. 1–17.

[16] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. "Exploration and exploitation in evolutionary algorithms: a survey". In: *ACM Computing Surveys (CSUR)* 45.3 (2013), p. 35.

[17] Kalyanmoy Deb et al. "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II". In: *International Conference on Parallel Problem Solving From Nature*. Springer. 2000, pp. 849–858.

[18] Juan J Durillo and Antonio J Nebro. "jMetal: A Java framework for multi-objective optimization". In: *Advances in Engineering Software* 42.10 (2011), pp. 760–771.

[19] Eclipse Foundation. *Eclipse Java development tools (JDT)*. 2018. URL: `EclipseJavadevelopmenttools`.

[20] Eclipse Foundation. *Eclipse Platform*. 2018. URL: `https://projects.eclipse.org/projects/eclipse.platform`.

[21] Marcela Genero, Mario Piattini, and Coral Calero. "Empirical validation of class diagram metrics". In: *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*. IEEE. 2002, pp. 195–203.

[22] Mark Harman et al. "Search based software engineering: Techniques, taxonomy, tutorial". In: *Empirical software engineering and verification*. Springer, 2012, pp. 1–59.

[23] Hao Hu et al. "Effective bug triage based on historical bug-fix information". In: *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*. IEEE. 2014, pp. 122–132.

[24] Paweł Janczarek and Janusz Sosnowski. "Investigating software testing and maintenance reports: Case study". In: *Information and Software Technology* 58 (2015), pp. 272–288.

[25] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. "Improving bug triage with bug tossing graphs". In: *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM. 2009, pp. 111–120.

[26] Dongwon Kang, Jinhwan Jung, and Doo-Hwan Bae. “Constraint-based human resource allocation in software projects”. In: *Software: Practice and Experience* 41.5 (2011), pp. 551–577.

[27] Muhammad Rezaul Karim et al. “An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer’s assignment to bugs”. In: *Journal of Software: Evolution and Process* 28.12 (2016), pp. 1025–1060.

[28] Elias Khalil, Mustafa Assaf, and Abdel Salam Sayyad. “Human resource optimization for bug fixing: balancing short-term and long-term objectives”. In: *International Symposium on Search Based Software Engineering*. Springer. 2017, pp. 124–129.

[29] A Güneş Koru et al. “An investigation into the functional form of the size-defect relationship for software modules”. In: *IEEE Transactions on Software Engineering* 35.2 (2009), pp. 293–304.

[30] Zhongpeng Lin et al. “An empirical study on bug assignment automation using Chinese bug data”. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2009, pp. 451–455.

[31] Sumio Masuda and Kazuo Nakajima. “An optimal algorithm for finding a maximum independent set of a circular-arc graph”. In: *SIAM Journal on Computing* 17.1 (1988), pp. 41–52.

[32] Andrew McCallum. “Multi-label text classification with a mixture model trained by EM”. In: *AAAI workshop on Text Learning*. 1999, pp. 1–7.

[33] G Murphy and D Cubranic. “Automatic bug triage using text categorization”. In: *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. 2004.

[34] Antonio J Nebro et al. “Mocell: A cellular genetic algorithm for multiobjective optimization”. In: *International Journal of Intelligent Systems* 24.7 (2009), pp. 726–746.

[35] Jihun Park et al. “Practical Human Resource Allocation in Software Projects Using Genetic Algorithm.” In: *SEKE*. 2014, pp. 688–694.

[36] H Pham. *Software Reliability, 2000.*

[37] Md Mainur Rahman and Guenther Ruhe. *Resource allocation and activity scheduling: bug fixing perspective*. Tech. rep. Technical Report, Software engineering decision support laboratory, University of Calgary, 2010.

[38] Md Mainur Rahmana et al. “An empirical investigation of a genetic algorithm for developer′s assignment to bugs”. In: *Proceedings of the First North American Search based Symposium*. 2012.

[39] Outi Räihä. “Applying genetic algorithms in software architecture design”. In: (2008).

[40] Jian Ren, Mark Harman, and Massimiliano Di Penta. "Cooperative co-evolutionary optimization of software project staff assignments and job scheduling". In: *International Symposium on Search Based Software Engineering*. Springer. 2011, pp. 127–141.

[41] Saad bin Saleem, Yijun Yu, and Bashar Nuseibeh. "An Empirical Study of Security Requirements in Planning Bug Fixes for an Open Source Software Project". In: *Technical Report No. 2012-01, The Open University* (2012).

[42] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. "On the value of user preferences in search-based software engineering: a case study in software product lines". In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 492–501.

[43] Chayanika Sharma, Sangeeta Sabharwal, and Ritu Sibal. "A survey on software testing techniques using genetic algorithm". In: *arXiv preprint arXiv:1411.1154* (2014).

[44] David Talby et al. "Agile software testing in a large-scale project". In: *IEEE software* 23.4 (2006), pp. 30–37.

[45] Gregory Tassey. "The economic impacts of inadequate infrastructure for software testing". In: *National Institute of Standards and Technology, RTI Project* 7007.011 (2002).

[46] Jin Wu and Shapour Azarm. "Metrics for quality assessment of a multiobjective design optimization solution set". In: *Journal of Mechanical Design* 123.1 (2001), pp. 18–25.

[47] Junchao Xiao and Wasif Afzal. "Search-based resource scheduling for bug fixing tasks". In: *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*. IEEE. 2010, pp. 133–142.

[48] Geunseok Yang, Tao Zhang, and Byungjeong Lee. "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports". In: *Computer software and applications conference (COMPSAC), 2014 IEEE 38th annual*. IEEE. 2014, pp. 97–106.

[49] Feng Zhang et al. "An empirical study on factors impacting bug fixing time". In: *2012 19th Working Conference on Reverse Engineering*. IEEE. 2012, pp. 225–234.

[50] Eckart Zitzler and Simon Künzli. "Indicator-based selection in multiobjective search". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2004, pp. 832–842.

[51] Eckart Zitzler and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271.

[52] Weiqin Zou et al. “Towards training set reduction for bug triage”. In: *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*. IEEE. 2011, pp. 576–581.

.1 Appendix A : Thesis External Links

Item	link
Eclipse Datasets	`https://sites.google.com/site/mrkarim/bug-data.zip?attredirects=0`
Industrial DataSets	`http://plan4bugs.me/industrial-data`
Experiment Page	`http://plan4bugs.me`
Experiment dataset	`http://plan4bugs.me/experiment-data`
Thesis Latex Document	https://www.overleaf.com/read/smgssbkchfhy
Thesis Source Code	`https://bitbucket.org/eliasdk/bugplanning`