2019

Multi-objective search-based approach for software project management

Wisam Haitham Abbood Al-Zubaidi

University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/theses1

Recommended Citation


Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
MULTI-OBJECTIVE SEARCH-BASED APPROACH FOR SOFTWARE PROJECT MANAGEMENT

A Thesis Submitted in Partial Fulfilment of the Requirements for the Award of the Degree of

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Wisam Haitham Abbood Al-Zubaidi
B.Sc.(Eng), M.Sc.(Eng)

School of Computing and Information Technology
Faculty of Engineering and Information Sciences

2019
I, Wisam Haitham Abbood AL-Zubaidi, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

(Signature Required)
Wisam Haitham Abbood AL-Zubaidi
31 March 2019
Dedicated to

my wife
my kids
and
my parents
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures/Illustrations</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>x</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Research questions</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Research main contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Structure of this thesis</td>
<td>7</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>9</td>
</tr>
<tr>
<td>2.1 Search-Based Software Engineering</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1 Metaheuristic Search-Based Techniques</td>
<td>12</td>
</tr>
<tr>
<td>2.1.2 Single Objective Evolutionary Algorithms</td>
<td>16</td>
</tr>
<tr>
<td>2.1.3 Multi-Objective Evolutionary Algorithms</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Machine Learning Algorithms</td>
<td>26</td>
</tr>
<tr>
<td>2.2.1 Case-Based Reasoning (CBR)</td>
<td>27</td>
</tr>
<tr>
<td>2.2.2 Random Forests (RF)</td>
<td>27</td>
</tr>
<tr>
<td>2.2.3 Linear Regression (LR)</td>
<td>28</td>
</tr>
<tr>
<td>2.3 Performance Metrics</td>
<td>30</td>
</tr>
<tr>
<td>2.4 Issue-Driven Software Projects Management</td>
<td>32</td>
</tr>
<tr>
<td>2.4.1 Issue Characteristics</td>
<td>33</td>
</tr>
<tr>
<td>2.4.2 Issue Lifecycle</td>
<td>35</td>
</tr>
<tr>
<td>2.4.3 Iteration Characteristics</td>
<td>36</td>
</tr>
<tr>
<td>2.4.4 Modern Code Review</td>
<td>38</td>
</tr>
<tr>
<td>2.5 Applications of the Search-Based Software Engineering</td>
<td>41</td>
</tr>
<tr>
<td>2.5.1 Software Requirements and Project Management</td>
<td>41</td>
</tr>
<tr>
<td>2.5.2 Software Analysis and Design</td>
<td>43</td>
</tr>
<tr>
<td>2.5.3 Software Testing</td>
<td>44</td>
</tr>
<tr>
<td>2.5.4 Software Maintenance</td>
<td>47</td>
</tr>
<tr>
<td>2.5.5 Other Applications</td>
<td>49</td>
</tr>
<tr>
<td>2.6 Chapter Summary</td>
<td>49</td>
</tr>
</tbody>
</table>
# Table of Contents

## 3 Iteration Planning
3.1 Motivation Example ........................................... 53
3.2 Approach .......................................................... 58
  3.2.1 Solution Representation ..................................... 58
  3.2.2 Fitness functions ............................................. 59
  3.2.3 Constraints .................................................. 62
  3.2.4 Evolutionary Search ......................................... 63
  3.2.5 Selecting a Solution From a Pareto Front ................. 64
3.3 Evaluation .......................................................... 66
  3.3.1 Datasets ...................................................... 67
  3.3.2 Experimental Settings and Measures ....................... 69
  3.3.3 Results ...................................................... 72
  3.3.4 Threats to Validity ......................................... 77
3.4 Related Work ...................................................... 78
3.5 Chapter Summary .................................................. 79

## 4 Issue effort and time estimation
4.1 Motivation example .............................................. 85
4.2 Issue features ..................................................... 87
  4.2.1 Title and Description ....................................... 87
  4.2.2 Issue Type .................................................. 88
  4.2.3 Priority ..................................................... 89
  4.2.4 Reporter .................................................... 89
  4.2.5 Components ................................................ 90
4.3 Approach ........................................................... 90
  4.3.1 Overview .................................................... 90
  4.3.2 Symbolic regression ....................................... 92
  4.3.3 Fitness functions ........................................... 94
  4.3.4 Evolutionary search ....................................... 96
4.4 Evaluation .......................................................... 101
  4.4.1 Datasets ...................................................... 104
  4.4.2 Experimental Settings and Measures ....................... 107
  4.4.3 Results ...................................................... 110
  4.4.4 Threats to validity ....................................... 124
4.5 Related work ...................................................... 125
4.6 Chapter summary ................................................. 129

## 5 Workload-Aware Code Reviewer Recommendation
5.1 Modern Code Review ............................................... 134
5.2 Search-based software reviewer recommendation ............ 137
  5.2.1 Approach Framework ........................................ 137
  5.2.2 Evolutionary search ....................................... 138
  5.2.3 Solution representation ................................... 140
  5.2.4 Reviewer Metrics .......................................... 141
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.5 Fitness functions</td>
<td>143</td>
</tr>
<tr>
<td>5.2.6 Selecting a solution from a Pareto front</td>
<td>146</td>
</tr>
<tr>
<td>5.3 Evaluation</td>
<td>148</td>
</tr>
<tr>
<td>5.3.1 Datasets</td>
<td>149</td>
</tr>
<tr>
<td>5.3.2 Experimental settings and measures</td>
<td>151</td>
</tr>
<tr>
<td>5.3.3 Results</td>
<td>154</td>
</tr>
<tr>
<td>5.3.4 Threats to validity</td>
<td>164</td>
</tr>
<tr>
<td>5.4 Related work</td>
<td>165</td>
</tr>
<tr>
<td>5.5 Chapter summary</td>
<td>168</td>
</tr>
<tr>
<td>6 Conclusions and future work</td>
<td>169</td>
</tr>
<tr>
<td>6.1 Summary of contributions</td>
<td>169</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>172</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Descriptive statistics of the sprints of the projects in our datasets</td>
<td>68</td>
</tr>
<tr>
<td>3.2</td>
<td>Evaluation results of MOSBIP (our approach using NSGA-II), Random Guessing (RG), the single objective approaches GA-GoA and GA-BV</td>
<td>74</td>
</tr>
<tr>
<td>3.3</td>
<td>Evaluation results for MOSBIP using different multi-objective optimization algorithms (Random Search, MOCell, and SPEA2)</td>
<td>75</td>
</tr>
<tr>
<td>3.4</td>
<td>Comparison of MOSBIP vs. RS, MOCell, and SPEA2 using Wilcoxon test and $A_{XY}$ effect size (in brackets)</td>
<td>76</td>
</tr>
<tr>
<td>3.5</td>
<td>Comparison of MOSBIP vs. GA-BV and GA-GoA using Wilcoxon test and $A_{XY}$ effect size (in brackets)</td>
<td>76</td>
</tr>
<tr>
<td>3.6</td>
<td>Comparison of MOSBIP against Random Guessing (RG) using Wilcoxon test and $A_{XY}$ effect size (in brackets)</td>
<td>77</td>
</tr>
<tr>
<td>4.1</td>
<td>Descriptive statistics of the resolution time for the projects in our datasets</td>
<td>106</td>
</tr>
<tr>
<td>4.2</td>
<td>Descriptive statistics of the story points for the projects in our datasets</td>
<td>106</td>
</tr>
<tr>
<td>4.3</td>
<td>Evaluation results of our approach MOIEE in terms of elapsed time, the baselines (Mean and Median) (the best results are highlighted in bold), MAE, MMRE and MdAE - the lower the better, SA - the higher the better</td>
<td>111</td>
</tr>
<tr>
<td>4.4</td>
<td>Evaluation results of our approach MOIEE in terms of story point estimation, the baselines (Mean and Median) (the best results are highlighted in bold), MAE, MMRE and MdAE - the lower the better, SA - the higher the better</td>
<td>112</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison between our approach MOIEE and the three baseline techniques in terms of story points using Wilcoxon test and $A_{XY}$ effect size (in brackets)</td>
<td>113</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison between our approach MOIEE and the three baseline techniques in terms of elapsed time using Wilcoxon test and $A_{XY}$ effect size (in brackets)</td>
<td>113</td>
</tr>
<tr>
<td>4.7</td>
<td>Evaluation results of our approach MOIEE in terms of elapsed time, the state-of-the-art techniques: Linear Regression (LR), Case-Based Reasoning (CBR), and Random Forests (RF). The results for the the single objective search with unlimited depth (GP-SAE) are also included – discussed later in RQ3 (the best results are highlighted in bold)</td>
<td>115</td>
</tr>
</tbody>
</table>
4.8 Evaluation results of our approach MOIEE in terms of story points, the state-of-the-art techniques: Linear Regression (LR), Case-Based Reasoning (CBR), and Random Forests (RF). The results for the single objective search with unlimited depth (GP-SAE) are also included discussed later in RQ4 (the best results are highlighted in bold). 116

4.9 A comparison of the MOIEE with both Choetkietikul et. al. and Porru et. al. using the Mean Absolute Error (MAE). 117

4.10 Comparison between our approach MOIEE and the state-of-the-art techniques (LR, CBR, RF, SMS-EMOA, and IBEA) in terms of elapsed time using Wilcoxon test and $A_{xy}$ effect size (in brackets). 117

4.11 Comparison between our approach MOIEE and the state-of-the-art techniques (LR, CBR, RF, SMS-EMOA, and IBEA) in terms of story points using Wilcoxon test and $A_{xy}$ effect size (in brackets). 117

4.12 Evaluation results for MOIEE in terms of story points using different multi-objective optimization algorithms (SMS-EMOA: S-metric selection evolutionary multi-objective algorithm, and IBEA: Indicator-based evolutionary algorithm). 118

4.13 Evaluation results for MOIEE in terms of elapsed time using different multi-objective optimization algorithms (SMS-EMOA: S-metric selection evolutionary multi-objective algorithm, and IBEA: Indicator-based evolutionary algorithm). 119

4.14 Comparison between our multi and single objective (MOIEE vs. GP-SAE) using Wilcoxon test and $A_{xy}$ effect size (in brackets) in terms of elapsed time. The second row reports the average tree size (i.e. the number of nodes) of a solution estimation model – the first number produced by MOIEE while the second number produced by GP-SAE). 121

4.15 Comparison between our multi and single objective (MOIEE vs. GP-SAE) using Wilcoxon test and $A_{xy}$ effect size (in brackets) in terms of story points. The second row reports the average tree size (i.e. the number of nodes) of a solution estimation model – the first number produced by MOIEE while the second number produced by GP-SAE). 122

4.16 MAE produced by our approach MOIEE in cross-project settings in terms of story points, source (trained project) and target (tested project). 123

5.1 A statistical summary of the studied datasets for each studied project. 150

5.2 Comparison of WLRRec vs. RS, MOCell, and SPEA2 using Wilcoxon test and $A_{xy}$ effect size (in brackets). 157

5.3 Comparison of WLRREC vs. GA-RevWL and GA-RevWL using Wilcoxon test and $A_{XY}$ effect size (in brackets). 163
# List of Figures

2.1 Search-based software engineering components (redrawn from Boussaid et al. [1]) ........................................ 11
2.2 Generic flowchart for GA ........................................ 16
2.3 An example of non-dominated fronts of the two objective optimization problem ........................................ 18
2.4 Generic flowchart IBEA procedure ........................................ 22
2.5 MOCell procedure ........................................ 23
2.6 The case-based reasoning (CBR) cycle ........................................ 28
2.7 An example of both regression and classification strategies of the RF ........................................ 29
2.8 Example of an issue recorded in JIRA ........................................ 35
2.9 Example of an issue workflow in jira ........................................ 36
2.10 Sprint Process Overview ........................................ 37
2.11 Example of a sprint in JBossDeveloper software ........................................ 38
2.12 An example of Gerrit code reviews in Open Stack project ........................................ 40
3.1 Example of a product backlog in JBoss ........................................ 54
3.2 Example of an issue ........................................ 55
3.3 Example of a sprint report of sprint *Thai Sprint 3* ........................................ 57
3.4 An example of how a candidate solution is represented using a bit string. ........................................ 59
3.5 An example of non-dominated fronts ........................................ 65
4.1 A Motivation example of an issue with estimated resolution time and story points. ........................................ 85
4.2 An overview of our approach ........................................ 91
4.3 An example of expression tree representing a candidate estimation model. Note that $f_i$ represents a feature of an issue. ........................................ 93
4.4 Non-dominated sorting genetic algorithm (NSGA-II) Flow chart ........................................ 99
4.5 An example of the mutation operator ........................................ 100
4.6 An example of the crossover operator ........................................ 100
4.7 An example of non-dominated fronts ........................................ 101
5.1 Motivation example from LibreOffice project ........................................ 136
5.2 An overview of our approach ........................................ 138
5.3 An example of non-dominated fronts ........................................ 139
5.4 An example of how a candidate solution is represented using a bit string .......................... 141
5.5 A graphical representation for the Pareto front and knee point ........................................ 147
5.6 Boxplots of results achieved by our WLRRec on the original dataset for the projects (Android, LibreOffice, Qt, OpenStack) in terms of precision recall, $f$-measure, and hypervolume .......................................................... 156
5.7 Boxplots of results achieved by WLRRec, RS, MOCell and SPEA2 for Android project ............ 158
5.8 Boxplots of results achieved by WLRRec, RS, MOCell and SPEA2 for LibreOffice project ............ 159
5.9 Boxplots of results achieved by WLRRec, RS, MOCell and SPEA2 for Qt project ...................... 160
5.10 Boxplots of results achieved by WLRRec, RS, MOCell and SPEA2 for OpenStack project ............ 161
5.11 Evaluation results of WLRRec, the single objective approaches GA-RevEXP and GA-RevWL .................. 162
List of Publications

Early versions of the work in this thesis were published as listed below:

- **Multi-objective iteration planning in agile development**

- **Multi-objective search-based approach to estimate issue resolution time**
MULTI-OBJECTIVE SEARCH-BASED APPROACH FOR SOFTWARE
PROJECT MANAGEMENT

Wisam Haitham Abbood AL-Zubaidi

A Thesis for Doctor of Philosophy
School of Computing and Information Technology
University of Wollongong

ABSTRACT

Project management covers the entire lifecycle of software, underpinning the success or failure of many software projects. Managing modern software projects often follows the incremental and iterative process where a software product is incrementally developed through a number of iterations. In each iteration, the development team needs to complete a number of issues, each of which can be implementing a new feature for the software, modifying an existing functionality, fixing a bug or conducting some other project tasks. Although this agile approach reduces the risk of project failures, managing projects at the level of issues and iterations is still highly difficult due to the inherent dynamic nature of software, especially in large-scale software projects. Challenges in this context can be in many forms such as making accurate estimations of the resolution time and effort of resolving issues or selecting suitable issues for upcoming iterations. These integral parts of planning is highly challenging since many factors need considering such as customer business value and the team’s historical estimations, capability and performance. Challenges also exist at the implementation level, such as managing the reviewing of code changes made to resolve issues. There is currently a serious lack of automated support which help project managers and software development teams address those challenges.

This thesis aims to fill those gaps. We leverage a huge amount of historical data in software projects to generate valuable insight for dealing with those challenges in managing iterations and issues. We reformulate those project management problems as search-based optimization problems and employ a range of evolutionary meta-heuristics search techniques to solve them. The search is simultaneously guided by a number of multiple fitness functions that express different objectives (e.g. customer business value, developer expertise and workload, and complexity of estimation models) and constraints (e.g. a team’s historical capability and performance) in the context of modern software projects. Using this approach, we build novel models for estimating issue resolution time and effort, suggesting appropriate issues for upcoming iterations in iteration planning and recommending suitable reviewers for code changes made to resolve issues. An extensive empirical evaluation on a range of large software projects (including Mesos, Usergrid, Aurora, Slider, Kylin, Mahout, Common, Hdfs, MapReduce, Yarn, Apstud, Mule, Dnn, Timob, Tisud, Xd, Nexus, Android, LibreOffice, Qt, and Openstack) demonstrates the highly effective performance of our approach against other alternative techniques (improvement between 1.83% to 550%) to show the effectiveness of our approach.

KEYWORDS: Iteration Planning, Agile Development, Effort Estimation
Acknowledgements

First and foremost, I would like to take this opportunity to extend my sincere gratitude and appreciation to my supervisor, Associate Professor Hoa Khanh Dam, for his continuous support, patient guidance, encouragement, and excellent motivation to complete my Ph.D. thesis. I must admit that I am truly fortunate for having worked with such a highly proficient and committed advisor. I have learned from him how to be a good scientist. Without his generous help and incredible support, it would never be possible for me to complete my thesis.

I would also like to thank my co-supervisor Prof. Aditya Ghose for his constant support and encouragements. He always gives constructive, insightful, helpful discussions, and valuable assistance.

I want to dedicate this work to my dearest people—my parents whom I am away from them now, and I do miss them so much. I am doing my best to make them happy and proud.

My deepest thanks to my lovely wife, Shaymaa, for all her dedicated support and kindness that deeply helped me to accomplish this work. I wish to dedicate this work to my kids our adorable Adam wonderful daughter Jumana. I would like to express my sincere thanks to my aunt Alia for her incredible help through the most difficult times.

I would like to thank my colleagues and friends in the Decision Systems Lab (DSL), It is an honour for me to be a part of the Decision Systems Lab.

It is a pleasure to thank those who made my degree possible, thank you all. God bless you all!
Chapter 1

Introduction

Software project management is critical to drive all software development activities ranging from requirements, design, implement to verification and validation, and testing. Project management has therefore significant impact on the outcomes of software projects. Delays and budget overrun have been a problem in many software projects. For example, the initial budget of NASA Check-out Launch Control System project was $200 million. This project has been cancelled because the allocated budget was overrun by another $200 million [2]. A recent study [3] found that in software projects with a budget of more than $15 million there was a schedule overrun (7%), cost overrun (45%), and 56% less delivered values than predicted. Most of these problems were attributed to the lack of effective project management in software development.

Nowadays, agile software development methods are widely practiced in the industry to manage software projects [4]. A modern software project often includes a number of iterations (e.g. sprints in Scrum) where the software product is developed through an iterative and incremental process. An iteration is typically a short period of fixed length (usually 2–4 weeks) [5]. In each iteration, the development team needs to complete a number of issues such as bugs, development tasks, and the requests for new features. This shifts the traditional model where all functionalities are delivered
in a single delivery to an agile, flexible model which involves a series of incremental deliveries and small iterations. There is however limited support for software project management at fine-grained levels of issues and iterations. This thesis proposes a multi-objective search-based evolutionary approach to provide decision-makers (e.g. team leaders and project managers) with various predictive support at different stages of software development. We formulate these prediction models as optimization problems and solve them using computing evolutionary techniques inspired by Darwin’s evolution theory. We leverage valuable insight from a huge amount of data generated from software development to build novel models for effort estimation and iteration planning in modern agile projects, and reviewers recommendation in the modern code review process.

Each agile project includes a product backlog which contains a list of things (e.g. user stories or issues) that need to be done within the project. Modern agile projects are organized in terms of iterations. Planning for those iterations is an important aspect of today’s software project management. Central to an iteration plan is a set of issues which the team has decided to complete during the iteration. Selecting issues from the product backlog for an upcoming iteration is a challenging task for agile teams. Product backlog can be large, and thus there are many potential selections to be considered. Currently, most agile teams heavily rely on experts’ subjective assessment \cite{6,7}, and there is a serious lack of automated support which can help the team to arrive at an optimal selection of issues. Project managers and other decision-makers would thus need advanced automated support for selecting issues during iteration planning.

When selecting issues for upcoming iterations, effort estimation is important for project managers and the development teams to understand how much effort and time would be required for resolving the selected issues. In modern agile projects, this effort can be measured in time or story points. Story points are relative values
which represent the effort involved in resolving an issue. Project managers need to estimate the effort for completing an issue in their project since these estimates are critical to the formation of their iteration plan. Substantial studies have been done on software effort estimation (e.g. [8], [9], [10], [11], [12], [13], [14], [15], [16]). They built the effort estimation models using metaheuristic evolutionary approaches such as genetic programming (GP) [9, 16], and non-dominated Sorting Genetic Algorithm (NSGA-II) [10, 14] for the whole projects. Little work has however been done for estimating issue time and effort using an evolutionary approach.

Resolving issues and completing project tasks often require modification to code in the form of patches. A common practice is that developers (patch authors) need to submit their patch which will be reviewed by other developers (code reviewers) before it is integrated into the main repositories [17]. Code review has been recognized as an essential stage for early defects identifications, aiming to improve quality in software projects [18, 19]. Code review is thus one of the important and efficient means in the software development process and has become an integral part of software project management. Part of this management task is finding the most suitable reviewers for the submitted patch. Recent work (e.g. [20], [21], [22], [23], [24], [25]–[27]) started addressing this challenging task, aiming to increase the review effectiveness and participation. To the best of our knowledge, none of them has leveraged multi-objective search-based approaches and considered both the reviewer’s expertise and workload in searching for suitable reviewers.

This thesis aims to fill all the gaps discussed earlier. We leverage a multi-objective search-based evolutionary approach and historical data from a large number of software engineering projects to build new models for estimating issue time and effort, selecting issues in iteration planning, and finding reviewers for code patches needed to resolve issues.
1.1 Research questions

Iteration planning is an important activity in managing software projects where the software team needs to decide what should be done (in terms of issues) for an upcoming iteration. In addition, this is a challenging task since the team may need to take into account several complex factors such as the goal of the iteration, priorities of the issues, and the effort of resolving issues and the team’s capability. The business value that a team delivers to the customers at the end of an iteration is also an important factor. Our first research question thus focuses on addressing this problem.

**Research question 1:** How to provide automated support for project managers and other decision-makers to select issues for an upcoming iteration during iteration planning?

Selecting issues for an iteration requires an accurate estimation of the effort and time needed to resolve the issues. Estimating the effort of resolving issues are important for development management. In addition, predicting how long they will be resolved is critical to manage customers and users’ expectation. Issue estimation thus can be measured in term of time (absolute measure) or story points (relative measure). Those estimations are used by different stakeholders as input for prioritizing issues, planning and scheduling for future iterations and releases, and tracking a team’s progress rate. The focus of our second research question is thus on this estimation support at the fine-grained level of issues.

**Research question 2:** How to develop an estimation model which provides highly accurate predictions on issue resolution time and effort in story points?

To resolve an issue, the development team often needs to make changes to the software system. These changes are usually in the form of code patches. It has become a common practice (especially in large software projects) that code patches must be reviewed before they are integrated into the main code line. Code review is thus an
essential part of the software development process. Finding the most suitable reviewers based on experience, related knowledge, and collaboration network has become an important task in managing software projects. Therefore, our last research question focuses on this aspect of software project management, considering both reviewers expertise and workload in a project.

**Research question 3:** *How to develop a model which recommends the most suitable reviewers for a code patch for resolving an issue using a multi-objective search-based approach?*

### 1.2 Research main contributions

We make use of historical data stored in issue-tracking systems (e.g. JIRA) to collect issue reports and iterations. We leverage data from well-known large open source repositories (e.g. Apache\(^1\)) and other popular open-source software systems (e.g. Android\(^2\), LibreOffice\(^3\), OpenStack\(^4\), and Qt\(^5\)). We have analyzed our data and investigated their characteristics in order to answer each research question. We then performed the pre-processing step (e.g. data cleaning) to build the datasets for our studies. The main objective of this thesis is to provide decision-makers with novel support at the early stages of the software development life cycle. To this end, we leveraged metaheuristic multi-objective search-based evolutionary algorithms. Extensive evaluations of our approach are performed where we compare the performance our approach against other alternative techniques to show the effectiveness of our approach over the existing models and techniques.

The main contributions of this thesis are outlined as follows:

1. **https://issues.apache.org/jira**
2. **https://source.android.com/**
3. **https://www.libreoffice.org/**
4. **https://www.openstack.org/**
5. **https://www.qt.io/**
1. We leverage a search-based software engineering (SBSE) approach to develop automated support for the team in selecting issues from the product backlog for an upcoming iteration. We have formulated this problem as a multi-objective optimization problem. Our approach iteratively generates candidate selections of issues for a given iteration, and searches for the optimal combination(s). The search is guided simultaneously by two objectives: maximizing the business value which the team delivers in the iteration while maximizing the alignment with regard to the iteration’s original goal. Our evaluation of 233 iterations from six large open source projects (MESOS, USERGRID, AURORA, SLIDER, KYLIN, and MAHOUT) demonstrates the effectiveness of our approach (Research question 1).

2. We develop accurate models for estimating issue resolution time and effort of each single issue in a software project. Using genetic programming (a metaheuristic optimization method), we iteratively generate candidate estimate models and search for the optimal estimation models. The search is guided by two objectives: maximizing the accuracy of the estimation model while minimizing its complexity (tree size). Our evaluation on 12,937 issues from 13 large open source projects (COMMON, HDFS, MAPREDUCE, YARN, MESOS, APSTUD, MULE, DNN, TIMOB, MESOS, TISTUD, XD, and NEXUS) demonstrates that our approach outperforms the baselines and state-of-the-art techniques in issue estimation (Research question 2).

3. We develop a novel framework which recommends suitable reviewers for a given code patch aiming to resolve an issue. Our model is able to identify the most appropriate reviewers for a submitted patch, taking into account both their expertise and workload using a multi-objective evolutionary approach. The search
explores two objectives: (1) maximizing the reviewer’s expertise; and (2) minimizing the workload difference between developers. An extensive evaluation on four large projects including Android, LibreOffice, Qt, and OpenStack, demonstrates that our approach is capable of recommending reviewers who have high reviewing experience with a low reviewing workload. Our recommendations aim to create a balancing workload across the developers while ensuring them be assigned to relevant patches. Our contribution advances the state-of-art of in code reviewer recommendations, helping improve the effectiveness and efficiency of the code review process. (Research question 3).

4. We have developed several large datasets for our studies. The datasets consist of iteration planning, issue effort and time estimation, and code reviewers recommendations extracted from large software projects such as MESOS, USERGRID, AURORA, SLIDER, KYLIN, MAHOUT, COMMON, HDFS, MAPREDUCE, YARN, APSTUD, MULE, DNN, TIMOB, TISTUD, XD, NEXUS, ANDROID, LIBREOFFICE, QT, and OPENSTACK. Our datasets for iteration planning and time estimation are the first dataset of its type. We have made our datasets publicly available for future research in those topics.

1.3 Structure of this thesis

This section details the general structure of this thesis. The remainder of this thesis is organized as follow:

- Chapter 2 outlines the fundamentals of the search based software engineering and provides background material of the search context for which this thesis is founded. It includes some background of search based software engineering

https://www.dropbox.com/sh/t7t9cjz593x1q14/AABz6P_3DGQgxQHdt1p2Xobqa?dl=0
1.3. **Structure of this thesis**

with the description of the search techniques, and also an overview of learning algorithms. This chapter also provides details about the issue-driven concept in the context of software open source projects, and also explains modern code review. In addition, this chapter presents the common applications of search-based software engineering.

- Chapter 3 presents a multi-objective iteration planning in agile development. This chapter describes our multi-objective approach to solving this problem by using evolutionary algorithms. We then discuss the experimental evaluation of our approach, where we elaborate on the dataset, the evaluation process and then report the results. The same structure has been followed in Chapter 4-5.

- Chapter 4 describes the estimation of issue resolution time and effort (story points) in software projects. We present our multi-objective approach to solving this problem by using evolutionary algorithms.

- Chapter 5 describes how our multi-objective search-based evolutionary approach generates a recommendation list of most appropriate reviewers to support decision-making in modern code review.

- Chapter 6 gives a summary of this thesis and discusses the possible future work.
Chapter 2

Background

This chapter briefly presents the fundamentals of the search based software engineering and provides an overview of the search context for which this thesis is founded. More precisely, the first Section 2.1 in this chapter illustrates some background of search based software engineering incorporating the description of the search techniques we used in this thesis. Section 2.2 gives an overview of the learning algorithms applied in this thesis. Then predictive performance metrics are covered in Section 2.3. In addition, the issue-driven concept in the context of software open source projects is detailed next in Section 2.4. We then explain modern code review (Section 2.4.4). The final Section 2.5 in this chapter describes the applications of the search-based software engineering technique in different software engineering areas.

2.1 Search-Based Software Engineering

In this section, we briefly present the necessary background of the search-based software engineering (SBSE) which is the core topic of this Ph.D. thesis. We briefly describe the metaheuristic search-based techniques in Section 2.1.1. We then explain the single objective evolutionary algorithms in Section 2.1.2 and the multi-objective
2.1. Search-Based Software Engineering

In general, for software development, solving problems in software engineering can be quite costly. Thus, a lot of effort has been made in order to automatically solve those problems, aiming to reduce the requirements of human resources and hence significantly reduce the development cost. In software engineering (SE), the main goal of using search-based software engineering (SBSE) practice is to convert SE problems into computational search problems (i.e. optimization problems), by applying a variety of computational metaheuristics search (ranging from single objective techniques to multi-objective techniques). In other words, reformulating problems to be framed as optimization problems to determine the maxima or minima of objective functions)\(^{28,29}\). Thereby, the solutions can be automatically generated and then evolved to obtain the optimal solution\(^{30}\).

In 2001, the SBSE had been proposed by Harman and Jones\(^{31}\), the authors were the first to apply this concept (SBSE), they expected to notice a significant development in this field. They have indicated that, in the near feature, there will be a growing and a measurable application of metaheuristic search in several aspects of the software engineering domain. Indeed, previously, SBSE approaches have been utilized for a wide range of different software engineering problems: requirements engineering\(^{32–36}\), project management\(^{37–43}\), bug fixing\(^{44}\), software testing\(^{45–47}\), model-driven software engineering\(^{48}\), software design\(^{49}\), software product Lines\(^{50–54}\), cloud computing\(^{55,56}\), source code and performance configuration optimization\(^{57–61}\), predictive modelling\(^{10,14,32,62–64}\), and refactoring\(^{65–70}\), among others.

Any SBSE approach requires software engineering problems to be defined in terms of three components (see Figure 2.1):

- Problem Representation: it is an essential step for a specific problem to be
2.1. Search-Based Software Engineering

reformulated into a search-based optimization problem. A solution (individual) needs to be encoded using discrete values (e.g. binary, integer, alphabet, tree, etc.). For ordering and sequencing problems, the solution can be represented as a permutation.

- Fitness Function: It is the backbone of SBSE. The search space might be very complex and include conflicting objectives. Besides, it is very challenging to enumerate all solutions within a reasonable time. However, a suitable fitness function can efficiently guide the search to achieve preferable solutions (i.e. near optimal). Fitness functions are thus used to compare if a solution is better than another one and they might be subjected to some of the constraints depending on the kinds of problems.

- Search-based Technique choice: the metaheuristic evolutionary algorithms (optimization strategy) investigates the search space, aiming to find the optimal solutions. To do so, search operators determine how solutions are different and then effectively can cross the search space. The generalization of the obtained results is highly related to the philosophy of the applied metaheuristic algorithms.

![Figure 2.1: Search-based software engineering components (redrawn from Bousaid et al.)](image-url)
2.1.1 Metaheuristic Search-Based Techniques

This section presents a brief overview of metaheuristic search-based techniques, especially those we have adopted in this thesis.

Metaheuristic is a Greek term that has been coined by Glover [73], “meta” refers to “beyond” and “to discover” means “heuristic”. The techniques used in this thesis are metaheuristics which are widely used to solved different optimization problems (e.g., [74–77]). The procedures of these techniques have an optimization perspective to explore in a search space and guide for the preferable solutions. The techniques are employed when the search (solution) space is extremely large in order to find proximate solutions (i.e. near-optimal solutions).

Metaheuristics are a nature-inspired (i.e. mimic the natural systems processes) [78–80], and can be classified into two types, population-based and others that do not work with populations [77,81]. Ramirez et al. [82] reported that metaheuristics evolutionary algorithms (EAs) are the most frequently (88%) adopted techniques. They have been the widely used population-based techniques (a majority of 81% of sources) amongst many currently available metaheuristics [82]. Of these algorithms, (57%) were genetic algorithms, (24%) for multi-objective evolutionary algorithms, and (5%) were reported for the genetic programming [82]. The focus of this thesis is mainly on the use of evolutionary algorithms both (single abjective and multi-objective) techniques. Next, we will describe the evolutionary algorithms, such type of algorithms apply what so-called the genetic operators (crossover, mutation, reproduction, and selection). Those operators are applied to evolve a set of individuals (i.e. population) which compete to survive regarding their fitness functions.

The basic operation of an evolutionary algorithm follows the notion of natural selection (i.e. Darwin’s principles). A view of evolutionary algorithm [83–85] is depicted in Algorithm 1. First, a population (i.e. a set of potential solutions) is randomly
generated. For a target problem to be solved, each individual solution includes all the decision variables (i.e. the information needed to select a more or less efficient solution) to this problem. For each solution, a fitness function (i.e. a measure of performance) needs to be defined. Such fitness function guides the search technique to find the preferable solution (i.e. compare if a solution is better than another one).

After that, based on the fitness function of each individual, a mechanism of selection is applied to decide the fittest individuals to be selected in the mating pool. Next, after the mating process, a set of offspring is generated through the variation operators (i.e. the combination of the parent and offspring to create a new generation). In this way, at the next generation, the selected of offspring will replace the parent population. The procedure is repeated until reaching the maximum number of generations. In the next section, we provide a brief description of genetic programming (GP), genetic algorithm (GA), and single-and multi-objective evolutionary algorithms that have been leveraged in this thesis.

**Algorithm 1** Evolutionary algorithm pseudo-code [83]

1: BEGIN
2: Random initialization of population
3: Fitness function evaluation for each solution
4: repeat
5: Parents selection ← reproduction step to select the fit solutions
6: Parents recombination ← combining parents information (mating population)
7: Offspring generation ← offspring generation by variation operators
8: Update population ← evaluation of new candidate solutions
9: Survivor selection ← individuals (solution) to be selected for the next generation
10: until the number of generations reached (Termination Condition)
11: END
2.1.1.1 Genetic Programming (GP)

As a branch of the genetic algorithm, genetic programming (GP) is an evolutionary approach designed to automatically evolve the computer programs in order to solve the computational problems [86–88]. In the real world, a number of successful GP applications have been well and widely known (e.g. [16, 89–99]) since Koza’s work in 1992 [100]. Similarly to the genetic algorithm, the strategy of this approach is also inspired by biological evolution. The representing solution is the main difference between the genetic algorithm and genetic programming. In genetic algorithms, a string of numbers is created to represent the solution while in genetic programming the solution (individual) is represented as a tree. This tree is built of two sets: the function set (internal node) and the terminal set (leaves/symbols). Both sets should contain the components which are appropriate for solving the target problem. For example, the function set includes (logic operators, arithmetic operators, mathematical functions, etc.), whereas the variables (i.e. features of the target problem) can be included in the terminal set. GP starts with a randomly generated initial population where trees are built of either fixed or variable depth (step1). Then, a fitness value is assigned for each individual (program) using the fitness function (step2). After that, based on the assigned fitness value, a selection of some individuals starts to form the parents and then the crossover and mutation are employed to generate new individuals hence using the fitness function to evaluate those new individuals (step3). Each individual in the newly created population is evaluated according to the fitness function (i.e. update the population) Step 4. The process in steps (2,3,4) is repeated till reaching maximum number of generation (i.e. stopping criteria) (step5).
2.1.1.2 Genetic Algorithm (GA)

Genetic algorithms are powerful optimization techniques that follow the natural selection (i.e. Darwin’s theory) to solve the optimization problems [101, 102, 102–105]. As a class of evolutionary algorithms (EAs), these algorithms argued as the most popular and widely used metaheuristic in many studies of search based software engineering [28, 106, 107]. GA considers four primary (basic) features: population, selection, crossover, and mutation. The goal of using GA is to find the fittest solution (individual) survived along all generations.

A basic genetic algorithm first, randomly generalizes the initial population which is then followed by the evaluation of the fitness function for each individual in this population. Then, the operators (selection, crossover, and mutation) are applied in a number of generations. In the selection operator, parents are chosen from the current population to generate the offspring based on the fitness value. Individuals with better fitness have much more chance to be selected. The commonly applied selection strategies are tournament selection and roulette-wheel selection. With this selection operator, crossover and mutation are the two essential steps applied to create the new generation. Two parents are combined to create the new offspring by crossover operator, this operator randomly picks crossover point(s) (single-point, two-point, multi-point) within parents where the interchanges happen. This operator works based on the predetermined crossover probability. The step of mutation is applied to keep the diversity (variation in selection) of population by randomly changing the solutions. This operator works with a very low level mutation probability. Once the three operators (selection, crossover, and mutation) are applied, each individual in the newly created population is evaluated according to the fitness function. This process is iteratively repeated till reaching the termination condition (i.e. maximum number of generation). The sequence of GA is given in Figure 2.2.
2.1. Search-Based Software Engineering

2.1.2 Single Objective Evolutionary Algorithms

A search technique is categorized depending on the kind of the optimization problem, more precisely, on the objectives numbers. A single type of optimization (i.e. single objective algorithm) needs to be used for the single objective problem. Hereby, the problem is described in terms of a unique objective (i.e. a single fitness function), and the algorithm is applied to one set of optimal solutions in order to gain the most optimal objective value (global optimum) [110,111]. For example, several studies [34,112] have used a variety of single objective optimization algorithms (e.g. simulated annealing algorithm, greedy algorithm) for next release problem (NRP) in order to consider requirements in next release planning to be delivered in the incremental software release process. The single objective optimization algorithms are population-based where

---

Figure 2.2: Generic flowchart for GA
the search process depends on the objective which needs to be maximized or minimized aiming to reach the most feasible solutions \[113\].

### 2.1.3 Multi-Objective Evolutionary Algorithms

The basic principles of the searching techniques for the multi-objective evolutionary algorithms (MOEAs) are similar to single objective evolutionary algorithms, but the major difference is the fitness evaluation method (i.e. the evaluation for the fitness function of a solution). The single objective algorithm is performed for a single objective function while in the multi-objective algorithm the fitness evaluation is for different objective functions where two or more conflictive objectives (i.e. the better in one objective, the worse in another one), can be optimized simultaneously. This optimization technique thus leads to a set of perto optimal nondominated solutions where the trade-off between the objectives is applied to determine the quality of a solution. The MOEAs are based on pareto dominance theory (i.e. the strength of the solutions dominance in the objective space) aiming to restrict the number (i.e population) of the generated solutions. Hence, the core task of the MOEAs is to evolve a set of nondominated solutions in order to reach the optimal Pareto front approximately \[108, 114, 126\].

The optimality of a solution in such strategy is determined using the notion of dominance. Hence, a solution on a Pareto front does not dominate another solution on the same front, i.e. the former is better than the latter with respect to at least one objective, and not worse in the other objective. For example, in Figure 2.3, $P_1$ solution does not dominate $P_2$ solution since the former is lower than the latter in objective 2 but is greater in objective 1. On the other hand, $P_3$ dominates $P_4$ since the $P_3$ has a lower objective 2 than $P_4$ and also has smaller objective 1.

We briefly describe the Multi-objective evolutionary algorithms used in this thesis
2.1. Search-Based Software Engineering

2.1.3.1 Non-Dominated Sorting Genetic Algorithm (NSGA-II)

As a search-based method, the NSGA-II was presented by Deb et al. [127]. Since then, it has been a well-known optimization algorithm that has been widely utilized in many search-based software engineering studies (e.g. project requirements and management [128–134], software design and analysis [135–138], software maintenance [139,140,140–147], software testing [148,150], amongst others). The main goal of this algorithm is to evolve the candidate solutions to gain and maintain a well-distributed set of near-optimal solutions, namely the Pareto front (non-dominated solutions) for solving a multi-objective optimization problem. The elitism of the non-dominated solution
provides a proper trade-off between all optimized objectives considering all objectives (i.e. without ignoring any objective). Hereby, in the non-dominance fronts, each solution is ranked regarding the level of non-domination (i.e. their dominance relation). This algorithm also follows the same basic principles of the genetic algorithms as the offspring is generated from the parents by applying the genetic operators. Although, the main difference is that, NSGA-II performs tournament selection based on both (dominance relation and the crowding distance) strategies.

NSGA-II starts to randomly generate an initial population $P_0$ in which each individual in the population is a candidate solution. The fitness values of each individual (solution) with respect to each fitness function are computed. The population is then undergone a selection process. The selected individuals form the parent to generate a new generation of individuals (i.e. offspring $Q_0$) through the crossover and mutation operators. Then a search for solutions that meet the optimized objectives is conducted. These solutions form a Pareto front of well-distributed set of near-optimal solutions. At each generation, NSGA-II sorts the current population into a number of non-dominated fronts (e.g. fronts 1, and 2 in Figure 2.3). Each non-dominated front contains individuals which do not dominate each other. Individuals in the first non-dominated front dominate those on the second front, and so on. Individuals in the same non-dominated front are assigned the same rank, which is the index of its front.

For example, in Figure 2.3, $P_1$ and $P_2$ have the same rank 1, while $P_4$ has rank 2. The crowding distance of each individual is then computed as the sum of the distance between itself and its nearest neighbours on the same front. The intuition here is that individuals with lesser domination rank are favoured in the case when they are on different fronts, and individuals in a less dense region (i.e. higher crowding distance) are preferred in the case when they are on the same front. This necessitates the pressure
for the population to move towards the Pareto Front and spread along it. The next generation is selected from a combination of the parent and offspring operation. In the final generation, NSGA-II returns a set of non-dominated solutions. This evolution process continues until a fixed number of generations has been reached (see Algorithm 2).

**Algorithm 2 NSGA-II procedure**

1: BEGIN
2: **Step 1:** randomly create an initial population $P_0$ of size $M$
3: **Step 2:** calculate the fitness values of each individual
4: **Step 3:** non-dominated ranking operation (sorting the population into different fronts)
5: **Step 4:** create an offspring $Q_0$ of size $M$ through (selection, crossover and mutation)
6: **Step 5:** evaluate the fitness of the new individuals
7: **Step 6:** merge parent and offspring population ($P_0 \cup Q_0$)
8: **Step 7:** sorting the current population into a number of non-dominated fronts
9: **Step 8:** compute the crowding distance of each individual, individuals with higher crowding distance are preferred.
10: **Step 9:** a combination of the parent and offspring to create next generation.
11: **Step 10:** evolution process continues until a fixed number of generations (stopping criterion) has been reached, if stopping criterion is not satisfied back to step 3.
12: END

### 2.1.3.2 Random Search (RS)

Random search algorithm is a baseline search technique that applied as a lowest benchmark to be compared against most of the search based metaheuristic algorithms $^{151}$. For any optimization problem, all applied metaheuristic algorithms should easily outperform the random search technique (i.e. a sanity check strategy) $^{31}$. Note that, random search is considered as a merely simple replacement technique where selection, crossover, and mutation are not applied. Thus, this algorithm cannot be categorized as an optimization algorithm. Given the same number of fitness values (evaluations), random search replaces the previous individual with a new one if the later has a better
fitness function. This changing procedure is performed repeatedly in the search space.

2.1.3.3 Indicator-Based Evolutionary Algorithm (IBEA)

This indicator-based evolutionary algorithm (IBEA) was suggested by Zitzler et al. \cite{152}. This algorithm has so far been applied in several studies \cite{153-160}. The main idea of this strategy is to formalize preferences which are in turn then called a performance indicator. The core function of this indicator is to compare the candidate solutions (in pairs) hence reflect the solutions' quality with respect to diversity and convergence. This algorithm (IBEA) is used for assigning fitness values according to the hypervolume estimation (i.e. hypervolume-based selection) as an indicator of the solutions in the current population in order to select the most elite solutions for the next population. This indicator (hypervolume) makes a contribution to the selection criterion, i.e., a solution with a lower (worse) hypervolume contribution is repeatedly removed from the current population aiming to reach the recommended size of elite solutions. IBEA starts with (step 1) randomly generating a population P. Then (step 2), for each solution from P the IBEA computes the fitness function. After that (step 3), this algorithm removes the individuals with the lowest fitness values by applying environmental selection process and update the remaining individuals by recalculating their fitness values. This procedure is repeated until reaching the recommended size of solutions (Step 4). Then the tournament selection operator is applied to the population to start the mating selection (step 5). Followed by the crossover and mutation operators to generate the offspring to be added to the population repeatedly until a fixed number of generations is reached (step 6). The IBEA procedure is illustrated in Figure 2.4.
2.1.3.4 Multiobjective Cellular Genetic Algorithm (MOCell)

The MOCell is another genetic algorithm which presented by Nebro et al. [161] to solve multi-objective optimization problems (e.g. [35, 162, 165]). This approach incorporates an external archive which includes the so far found non-dominated solutions. The execution of this algorithm is NSGA-II based strategy. That way, the archive uses the crowding distance to sustain the Pareto front diversity. During the crowding distance strategy, after each generation, the MOCell applies the feedback process to
remove a number of solutions (individuals) from the archive back into the population. It thus replaces the randomly selected existing individuals in the population. MOCell is a cellular type of the genetic algorithm where a corporation can only occur among individuals in a nearby neighbor.

MOCell begins with creating an empty Pareto front. Individuals are selected, and then crossover and mutation are applied. Then, two parents are selected for each solution from the nearby neighborhood. The algorithm then re-combines them to create an offspring. After that, the resulting individual (offspring) needs to be evaluated and then inserted in both main and external archive populations to replace the individual which is dominated by the offspring (i.e., the individual with worse crowding distance). The obtained non-dominated individual is then inserted in the Pareto front. Afterwards, each solution (individual) in the archive is again to be ranked according to the crowding distance. Thus, in case the Pareto front is full, the algorithm will remove the solution of the worst crowding distance value. Finally, the feedback procedure is applied to remove individuals from the archive back into the population. The overall procedure is repeated till reaching the termination condition (stopping criteria). Figure 2.5 shows the MOCell procedure.

Figure 2.5: MOCell procedure (redrawn from Nebro et al. [161])
2.1. Search-Based Software Engineering

2.1.3.5  S-metric Selection Evolutionary Multi-Objective Algorithm (SMS-EMOA)

The SMS-EMOA is a steady-state algorithm was firstly introduced by Beume et al. [166]. Later on, there was an increasing amount of studies that have adopted this algorithm successfully [167–176]. This strategy is applied to constant population size, and it generates only one new individual in each iteration. It thus updates population members (individuals) within a steady-state strategy. Similarly to the IBEA, SMS-EMOA is also hypervolume indicator-based algorithm. However, this algorithm is designed to combine indicator-based selection (selection mechanism) with the non-dominated sorting (i.e. NSGA-II based for ranking criterion), aiming to maintain diversity and convergence.

The SMS-EMOA starts with (step 1) generating only one offspring individual (i.e. solution by iteration) from an initial population by the randomised variation operators (i.e. mutation and crossover). Afterward (step 2), to select the individual for the next population, the algorithm applies both the Pareto ranking and hypervolume. Then (step 3), it combines the generated new individual with the current population to gain the next population. The non-dominated sorting technique is then employed to divide the next population and hence to obtain diverse non-domination levels (each level is called a front) (step 4). After that (step 5), the selection process is applied to decide which solution will be removed where the hypervolume is used to remove individuals with the worst-ranked front. This procedure is repeated until the recommended population size is reached (stopping criteria), see Algorithm 3.

2.1.3.6  Strength Pareto Evolutionary Algorithm-II (SPEA2)

The SPEA2 is another multi-objective evolutionary algorithm that was proposed by Zitzler et al. [177], and has been widely employed by several researchers [178–185].
Algorithm 3 SMS-EMOA pseudo-code

1: BEGIN
2: **Step 1: Initialization**
3: \( P_0 \leftarrow \) create initial population with size \( S \)
4: **Step 2: Looping**
5: repeat
6: Solution by iteration \( \leftarrow \) generating only one new individual (offspring \( Q \)) from an initial population by the randomised variation operators (crossover and mutation)
7: Select the individual from \( P_0 \) for the next population
8: Create the next population \( N_p \leftarrow Q \cup P_0 \)
9: Apply the fast-nondominated-sorting \( \leftarrow \) divide the \( N_p \) to obtain different non-domination levels
10: Selection process \( \leftarrow \) hypervolume used to remove individuals with the worst-ranked front
11: until the number of generations reached (Stopping criteria)
12: END

This approach uses two populations (main and the external archive). This external archive population (external non-dominated set) includes the non-dominated solutions (individuals) that survive along the evolutionary process (i.e. the fittest individuals found). Besides, a strength value needs to be computed for each solution in this archive. There is a proportional relation between this strength value and the solutions number that dominated by a certain solution. The SPEA2 algorithm applies the strategy of the Pareto-based fitness assignment by which the fitness of each individual in the current population needs to be evaluated based on the strength values of all external non-dominated set.

Initially, SPEA2 randomly generate the population whereas the archive is empty. Then, the archive is filled with non-dominated solutions from the population (i.e. solutions with better fitness). After that, the algorithm deletes the dominated solutions from the archive. The fitness value of each solution in both population and archive is then assigned. The binary tournament selection and replacement are then applied. A new population is generated after applying the genetic operator. Hence,
this evolutionary process is stopped if it meets the stopping conditions. Otherwise, the 
non-dominated solutions from the initial population are to be copied to the archive. 
If the number of solutions in the archive is less than the number of non-dominated 
solutions, the operator of truncation is applied, and the whole evolutionary process is 
repeated (see Algorithm 4).

Algorithm 4 SPEA2 algorithm \cite{120,177}

\begin{enumerate}
\item BEGIN
\item Randomly generate an initial population $P_0$ and empty archive $A_0$
\item \textbf{while} the number of generations (stopping criteria) not reached \textbf{do}
\item \quad Fitness evaluation $\leftarrow$ compute the fitness values for both population and 
archive
\item \quad Create new archive $\leftarrow$ copy the non-dominated solutions from both archive $A_0$
and population $P_0$
\item \quad Apply truncation operator $\leftarrow$ to delete solution from $A_0$ if solution number 
exceeds the archive capacity; if not then fill the archive from population $P_0$ (non 
dominated set of solutions)
\item \quad Re-combine $\leftarrow P_0 \cup A_0$
\item \quad Create offspring $\leftarrow$ by tournament selection and then apply crossover and 
mutation
\item \quad Evaluate fitness values for the new population
\item \textbf{end while}
\item END
\end{enumerate}

2.2 Machine Learning Algorithms

In this section, we briefly describe machine learning algorithms which are well-known 
and have been commonly used in software engineering. It includes the basic infor-
mation regarding the algorithms that we have used as baselines in this thesis: the 
linear regression (LR) in Section 2.2.3, random forests (RF) in (Section 2.2.2), and 
case-based reasoning (CBR) (in Section 2.2.1).
2.2.1 Case-Based Reasoning (CBR)

CBR is an artificial intelligence method, it is one of the widely used machine learning algorithms in the software engineering field [186-192]. The CBR system differs from other learning algorithms in that the approach is directly applied on the repository of past cases (cased-based), then it solves the problem based on the similarity of those cases (i.e. CBR does not require an aspect of “identical previous problem”). Hence, the model is expressed through those adapted cases (i.e. the CBR is not model based).

The strategy of the CBR cycle includes four general steps as describe by Aamodt et al. [193]: 1) Retrieve: CBR system searches for the cases that are almost similar to the objective problem. 2) Reuse: applying the previously conducted actions (from past problems) for solving a new problem. 3) Revise: the newly presented solution is to be revised for the new problem and then validate it against the case base. 4) Retain: after validating (adjustment) the results, the new case is inserted into the case base (i.e. retain current experience to solve the future problems), see Figure 2.6.

For case-based reasoning (CBR), we used the popular k-nearest-neighbor (KNN) algorithm. KNN is a simple, precise, and effective technique that has been frequently applied in both classification and regression cases (e.g. [14, 194, 195]). In both cases, the choice of K is very crucial as it results in better performance of the KNN algorithm and determines whether it will be used for classification or regression. The selection of the similarity metric is another factor that has an impact on the KNN performance. For example, the Euclidean distance which is used to measure the distance among instances.

2.2.2 Random Forests (RF)

Random Forests (RF) is a flexible supervised learning algorithm used for classification and regression tasks (i.e. an ensemble learning method). This algorithm is proposed
2.2. Machine Learning Algorithms

Figure 2.6: The case-based reasoning (CBR) cycle (redrawn from Aamodt et al. [193] and Mantaras et al. [196])

by Breiman in 2001 [197] as an effective predictive modeling tool. The simplicity of this approach has made it one of the common and best used machine learning algorithms in a wide range of problems [198–203]. Random forest is an improved decision tree approach, which builds many decision trees (classification) to be combined in order to attain and maintain a better stable and accurate prediction. At the training time, random forest selects one of the constructed decision trees as a final model. That way, the selected model is the one that the maximum number of votes (by decision trees) are for classification. While in regression, RF obtains the average predictions for each individual decision tree. Figure shows an example of both regression and classification strategies of the RF.

2.2.3 Linear Regression (LR)

This approach is utilized to modeling the relationship between variables (independent and dependent) to the observed data [204–206]. LR is a prediction method that
2.2. Machine Learning Algorithms

![Diagram of Machine Learning Algorithms](image)

**Figure 2.7:** An example of both regression and classification strategies of the RF represents straight line (linear equation) through the two variables: dependent denoted by \(Y\) and independent by \(X\). There are two types of the linear regression modeling based on the number of independent variables: the simple linear regression model (one independent variable in the model), while multiple linear regression model includes multiple independent variables. The model of simple linear regression is represented in the formula below \([207]\):

\[
Y = \beta_0 + \beta_1 X + \varepsilon
\]  

(2.1)

Where variable \(Y\) is dependent, and variable \(X\) is independent. The coefficients are \(\beta_0\) and \(\beta_1\), and \(\varepsilon\) is a random error.
2.3 Performance Metrics

In this section, we focus on some of the common performance measures employed in our studies to assess predictive performance. We briefly discuss: 1) information retrieval metrics (precision, recall, F-measure), to assess the performance of the recommendation system; 2) regression metrics (MAE, MMRE, MdAE, SA), to measure the error between the estimated target values and the actual target values (i.e. ground truth); 3) multi-objective evaluation metric (Hypervolume), this is a quality indicator for multi-objective optimization performance. The performance metrics are described as follows:

- **Precision (Prec):** The ratio of the correctly recommended document over all the recommended document. It is calculated as:

  \[ \text{Prec} = \frac{|\text{Actual} \cap \text{Recommended}|}{|\text{Recommended}|} \quad (2.2) \]

- **Recall (Re):** The ratio of the correctly recommended document over all the actual completed document. It is calculated as:

  \[ \text{Re} = \frac{|\text{Actual} \cap \text{Recommended}|}{|\text{Actual}|} \quad (2.3) \]

- **F-measure (F1):** The weighted harmonic average of the recall and precision. It is calculated as:

  \[ F1 = \frac{2 \times (\text{Prec} \times \text{Re})}{(\text{Prec} + \text{Re})} \quad (2.4) \]

- **Mean Absolute Error (MAE):** This measure reflects the average of all abso-
lute errors. It is calculated as:

\[ MAE = \frac{1}{N} \sum_{i=1}^{N} |Actual - Estimated| \]  \hspace{1cm} (2.5)

- **Mean of the Magnitude of Relative Error (MMRE):** This measure reflects the mean value of the Magnitude of Relative Error (MRE) across all estimated values and the actual values. It is calculated as:

\[ MMRE = \frac{1}{N} \sum_{i=1}^{N} \frac{|Actual - Estimated|}{Actual} \]  \hspace{1cm} (2.6)

Where \( N \) is the samples’ number in a test set, \( Actual \) is the actual value, and \( Estimated \) is the estimated value.

- **Median absolute error (MdAE):** The median of the absolute errors between the actual value and the estimated value. It is calculated as:

\[ MdAE = Median\{|Actual - Estimated|\} \]  \hspace{1cm} (2.7)

- **Standardized Accuracy (SA):** It assesses how good an estimation model is with respect to random guessing. It is calculated as [208]:

\[ SA = (1 - \frac{MAE}{MAE_{\text{guess}}}) \times 100 \]  \hspace{1cm} (2.8)

where \( MAE_{\text{guess}} \) is the \( MAE \) averaging a large number of random guesses. Estimates with larger SA are more useful.

- **Hypervolume (HV):** Hypervolume is a quality indicator for multi-objective optimization. It is used to measure the volume of the space covered by the non-dominated solutions. The higher the hypervolume, the better the performance.
2.4. Issue-Driven Software Projects Management

Hypervolume is the only metric that has the capability to consider all three aspects (diversity, cardinality, and accuracy) \[177,209\]. It reflects the convergence and diversity of the solutions on a Pareto front. It is calculated as \[160,210\]:

\[
HV = \text{volume} \left( \bigcup_{i=1}^{S} v_i \right)
\] (2.9)

Where \( S \) is a set of solutions from the Pareto front to be assessed and \( v_i \) is the hypercube space established between each solution \( i \) and distance (reference) point by all solutions.

2.4 Issue-Driven Software Projects Management

In the previous sections, we discussed the background of search based software engineering algorithms, and learning algorithms. In this section, we provide an overview of issue-driven concept in the context of software projects management. It includes the essential basic information regarding an issue characteristics recorded in issue tracking system in Section \(2.4.1\), and also briefly illustrates the life cycle of an issue in Section \(2.4.2\). We then explain how an issue characteristic supports the developing of an iteration for the open source projects in an issue tracking system.

Most of today’s software projects are issue-driven where a project consists of a number of past issues (i.e. issues that have been closed), ongoing issues (i.e. issues that the team is working on), and new issues (i.e. issues that have just been created). The end-users may want to know when the new functionality they requested will be implemented. Thereby, knowing when an issue will be resolved is extremely important for many stakeholders.

An issue tracking system is highly important to provide a better and successful practice in the development of open source projects. Such a system (e.g. JIRA) is
quite valuable software tool as it provides team members with a central place where the process of project’s development is visible to all team members so that they can know their current tasks and decide what to do next \cite{211}. In an issue tracking system, team members can manage issues by specifying each task in relation to the recorded issue \cite{212}. This system records what has been performed on an issue (i.e. team’s actions) in the form of comments so that all team members are capable of reviewing and tracking the progress of their tasks on each single issue (i.e. increase the shared accountability).

### 2.4.1 Issue Characteristics

Broadly, the term issue indicates bugs, development tasks, and the requests for new features \cite{213,215}. Issues can be categories (created) by stockholders (end-users, developers, or the persons who involved in the quality assurance task, i.e., code reviewers, and managers). In software projects, an issue tracking system is the tool that teams (developers) use to keep tracking the progress of the enhancement process by collecting the feedback information regarding the defects that occur in released systems. For example, common issue tracking systems include Bugzilla\footnote{https://www.bugzilla.org/}, Jira, and among others. These systems assist the software development teams to review (monitor) issue’s status, to assign issues to developers, and to decide what to do next (i.e. planned release). An individual record in an issue tracking system is known as an issue/ problem report (ticket) \cite{216}.

In a software system, large open source projects such as Apache, Mulesoft, and JBoss have plenty of issues collected every day. Intuitively, around 65 to 89 issue reports have been daily collected by Eclipse and Firefox respectively \cite{215}. Another study also showed 170 and 120 new issue reports have been daily received by Mozilla.
and Eclipse projects respectively [217]. Usually, the number of received issues is greater than the development team size. That is why team leaders need to assign them to developers based on an issue priority, and the field severity [218].

In the issue tracking system, the recorded issue has a number of attributes such as type, priority, a textual description (summary and description), status, fix version, assign or reporter, resolution, and due date. Those attributes are essential to describe the characteristics of each single issue. The type of an issue indicates the nature of an issue (e.g. new feature, defect) where the textual description provides the detailed nature of the issue. The priority of an issue reflects its importance from the client’s perspective (e.g. new functionality urgently needed or a critical bug which must be fixed as soon as possible). The assignee is the person who is an issue assigned to in order to be fixed. The status specifies an issue’s current state in it’s life cycle in Section 2.4.2. The fix version is the field that designates each issue release version (the higher number of fix versions, the more attention will be need regarding testing and developing). The resolution is an issue record that to be set by the developers while reviewing the issue. It also describes the way that an issue has been completed (i.e. an issue reported in an open or closed state). The developers mark an issue with an open status (unconfirmed, In progress, confirmed) when the solution is found yet. An issue is described closed (done, resolved, fixed) when the solution is found. The due date gives the time about when an issue will be resolved. An example of issue ID Hadoop HDFS / HDFS-12578 with its attributes reported in the issue tracking system (jira) is shown in Figure 2.8.

\[2\]https://issues.apache.org/jira/browse/HDFS-12578
2.4. Issue-Driven Software Projects Management

2.4.2 Issue Lifecycle

The issue life cycle is an essential aspect where an issue goes through several steps of the resolution process. Once an issue is created, it passes through a defined lifecycle (i.e. a set of state attribute and transitions) which is also called an issue workflow (i.e. development cycles). This process starts at the time an issue is reported and finishes when this issue is closed. After an issue being discovered, an issue report is submitted to be then assigned to developers in order to be fixed (resolved) hence verified and closed. Note that, lifecycles are different as each type of issue has its own lifecycle that can differ from project to project. For example, Figure 2.9 depicts the issue lifecycle in the JIRA’s system workflow. Jira provides a generic set of standard workflow steps. There are five possible states which issues pass through (e.g. Open/Reopened, InProgress, Resolved, Closed). Once a new issue is created, developers conduct issue investigation (i.e. triage process) to confirm the correctness of the recorded issue report. Then, all important issue’s states (attributes) are determined. Open state is

3https://confluence.atlassian.com/display/JIRA052/Configuring+Workflow
2.4. Issue-Driven Software Projects Management

set when an issue is in the initial. Then it passes through a state of InProgress where the issue is received by an assignee (developer) to be worked on (fixed/resolved). When a resolution is taken (i.e. developer finishes the work on the issue), another developer (reporter) receives the issue for a waiting verification. The issue then is reported as either reopened (i.e. incorrect resolution) and marked as verified, or closed to be finally marked as a closed state (i.e. fixed or correct resolution).

![Diagram of issue workflow in JIRA](https://confluence.atlassian.com/display/JIRA052/Configuring+Workflow)

**Figure 2.9:** Example of an issue workflow in JIRA, adopted from https://confluence.atlassian.com/display/JIRA052/Configuring+Workflow

2.4.3 Iteration Characteristics

A software project includes a number of iterations with fixed-length (e.g. sprints in Scrum). An iteration is typically a short period (usually 2–4 weeks) in which the development team works on software products to design, implement, test and deliver a distinct product increment. In the industry, Scrum is one of the popular and widely used software development methodologies. An overview of the sprint process (Scrum methodology) is depicted in Figure 2.10.
2.4. Issue-Driven Software Projects Management

The basis of the Scrum method is to develop software by the incremental and iterative process through repeated cycles and in smaller parts at a time. In each iteration (sprint), a development team needs to complete a number of user stories and/or tasks, which are usually selected from the product backlog and recorded as issues in an issue tracking system (e.g., JIRA). Issues in the backlog are prioritized, which reflects the urgency and importance of the tasks from the client’s (e.g., product owner) perspective. Then, sprints can be created by the development team from the selected list of the prioritized issues.

An example of Sprint 8th in JBossDeveloper dashboard of the source BuildTracker Agile Board is displayed in Figure 2.11. The sprint 8th includes 8 issues divided into 4 states: To Do, In Progress, Pull Request Sent, and Done, by the Scrum progress monitoring and they represent an issue states in its lifecycle. To Do is set when issues are in the opened state. Issues then changed to be moved from To Do state into InProgress state (i.e., development progress). After that, a pull request is created when a developer demands that the work needs to be reviewed by another developer and hence merge the changes in. Consequently, the final state of Done thus includes the issues with the closed state. When the sprint ends, a report of completed, in-

---

**Figure 2.10: Sprint Process Overview**
completed, and removed issues is recorded. Hence, the Scrum team can provide a consistent delivery capability as the team keeps tracking and monitoring their tasks executions and progresses (i.e. development and productivity).

![Figure 2.11: Example of a sprint in JBossDeveloper software](image)

### 2.4.4 Modern Code Review

Resolving issues often requires changes to code. Code review is one of the important and efficient means in the development of software quality. The core motivation of the code review process is to early identify and reduce defects in code change (e.g., time-consuming, costs and project sustainability) aiming to control quality in software projects \[18, 19\]. In this process, developers (patch authors) submit their code change to be peer reviewed by other developers (code reviewers) and hence to be incorporated into the system (main repositories) \[17\]. Traditionally, manual inspection of code change is the most accepted industrial practice \[19\]. However, such practice is time-consuming especially when developers work from different time zones and locations.
2.4. Issue-Driven Software Projects Management

Currently, the modern software development teams apply Modern Code review (MCR) which is a tool-based review methodology that provides a lightweight and automated techniques to facilitate and support the code review process. For this purpose, the developers adopt exclusively dedicated code review tools to manage the MCR process (e.g., Gerrit which is a web-based review tool). These tools allow developers to submit patches and to choose relevant reviewers to inspect their patches. Broadly speaking, a developer (i.e. patch author) invites a set of code reviewers to identify the weakness of the submitted new patch (code change) and hence to discuss and suggest fixes. Then, when one or more code reviewers approve the code change, the change will be integrated into main software repositories.

The MCR practice mainly emphasizes on team members collaboration aiming to attain and maintain a high quality of software products. Team collaboration can bring more benefits to software developers regarding increasing the awareness and knowledge transfer. Experience and knowledge have been explicitly discussed as crucial elements for the successfulness of the review process (i.e. reviewers with prior knowledge in term of code and context can quickly provide valuable feedback to the author) [219,220]. In software projects, reviewers’ expertise with the changed files and review collaboration are the main factors that a patch author considers when inviting code reviewers [25,220]. However, due to the informal nature of the lightweight MCR process, it may often reduce the number of the participated reviewers. Consequently, that would negatively impact the software quality and reviewing timeliness due to an insufficient amount of participation and discussion between developers (patch authors and reviewers).

The impact of number of participated reviewers has been discussed in several studies, for example, Thongtanunam et al. [221] investigated the relation between the characteristics of patches and the amount of review participation. They found that the amount of review participation in the past can be used as a significant indicator
of the quality of review participation. McIntosh et al. [222] and Thongtanunam et al. [21] found that a lack of review participation is highly related to the long-term negative impact on software quality. Kononenko et al. [20] also found that the number of involved reviewers has an effect on the quality of the code review process. Hence, reviewer participation can be described as the main challenge in MCR process [23, 223, 224]. The lightweight MCR has been adopted in many open source projects (e.g., Android, Qt, LibreOffice, OpenStack). Figure 1 shows an example code review in Open Stack project which was uploaded on the 29th of January 2019. The figure is used to describe the typical review process in the Gerrit interface of the patch #633564.

![An example code review in Open Stack project](https://review.openstack.org/#/c/633564/)

**Figure 2.12:** An example of Gerrit code reviews in Open Stack project

---

4https://review.openstack.org/#/c/633564/
2.5 Applications of the Search-Based Software Engineering

Search-based software engineering (SBSE) refers to a metaheuristic optimization techniques. These techniques seek to transform the software engineering problem into a search (optimization) problem to be evolved, aiming to select the near-optimal solutions [225]. SBSE is widely applicable to all steps in the process of the software development cycle. In this section, we discuss the applications of this search technique in different software engineering areas. The discussion, however, does not comprehensively cover all software engineering activities. We thus focus on providing a broad scope of the SBSE applications in different areas:

2.5.1 Software Requirements and Project Management

Software requirements process is the early stage in the software development life cycle where requirements development systematically pass through a number of steps to understand the problem (e.g. analyzing, documenting, and reviewing) [226]. This process determines and manages the needs of the software’s users (stakeholders). The goal of requirements phase is to consider multiple criteria and hence satisfy the diverse interests of stakeholders by helping software engineers in the decision-making process. Software engineers need to decide which requirement should be implemented in the next release version (determine the priority of requirements) and then to maximize the delivered software product value [34]. Within the software requirements stage, the SBSE methods have been applied for the requirements selection and optimization in order to find a subset with the best requirements to be included in the next release (e.g. the popularly called Next Release Problem (NRP) [36]) to meet stakeholders different requests and constraints. For example, several work (e.g. [133, 134, 227])
addressed multiple objectives Next Release Problem (NRP). Finkelstein et al. [133] considered what is the next set of requirements based on each customer’s idea to satisfy all stakeholders (i.e. fulfilled requirements among multiple customers), while Zhang et al. [134] dealt with stakeholders satisfaction by considering each stakeholder as an independent objective (i.e. single choice of requirements). Harman et al. [227] used dynamic programming to formulate an optimization approach considering both the complexity and size of NRP by dealing with each objective as a separate objective function to solve this problem.

Regarding the project management, SBSE has been employed for project scheduling (i.e. resource planning and task scheduling), and predictive modeling. These activities are essential to achieving a successful project [41]. One of the problems in the project scheduling is the portfolio selection. Kremmel et al. [228] applied multi-objective algorithm namely, mPOEMS. This approach considered five decision factors as objectives: project revenue, resource distribution, the strategy of the selected project, involved risk, and project interdependencies. The approach showed the ability to optimize those objectives in terms of both project selection and scheduling. Rodriguez et al. [43] worked on another planning task. They applied NSGA-II approach considering the cost, time and productivity to be optimized. The approach helped software project managers to find the better values regarding scheduling estimates and team size. Another study [229] has worked on staff to tasks allocation problem within search-based project scheduling. This work applied the ε-MOEA approach to minimize project duration and cost aiming to find potent project schedules. Recently, Shen et al. [230] adapted the multi-objective Two-Archive memetic algorithm. The algorithm was described as proactive approach in solving rescheduling problem. It considered the employees’ satisfaction as an objective in addition to project cost, duration, stability, and robustness. This approach addressed the human-based factors
and hence helped software engineers in making better decisions.

An accurate software effort prediction is very necessary [231]. Harman [63] argued the close connections (relationship) regarding challenges encountered by SBSE techniques and the predictive modeling. Cost estimation is a very demanding task area where search based predictive modeling technique has been widely explored. Software project managers are required to provide cost and effort estimation early in the development life cycle of software engineering in order to achieve a successful software project management [232]. For example, Dolado [233] has applied genetic programming (GP) for project cost estimation in relation to the observed project effort. Another study [9] estimated the software project effort. They examined the application of different approaches including genetic programming to produce better solutions regarding effort estimation. Afzal et al. [62] investigated the benefit of symbolic regression using GP model for prediction and estimation in software engineering. The results supported using GP in the software system for quality classification, effort estimation, and fault prediction.

2.5.2 Software Analysis and Design

The main focus in the areas of search-based software analysis and design is the automation of different tasks for different models [49]. Mostly, multi-objective approaches have been applied in the area of software product lines (SPL). A Software Product Line (SPL) [52, 53, 234, 235] is a group of correlated software products which distinguished by having some shared core functionality. Each member in this group differs in some particular features (attributes) provided by each product. These differences assist the product line in finding the variability demanded by different platforms and different users. As a design task, the main purpose of the SPL method is to obtain an optimal solution by selecting the features that meet stakeholders requirements (i.e.
2.5. Applications of the Search-Based Software Engineering

SPL optimization problem).

Typically, the software product lines comprise a large number of attributes that are merged in complex relations which results in a big number of individual software systems [53]. These systems need to be designed, managed and then implemented efficiently and effectively. SPL related problems are thus appropriate for the application of search-based software engineering techniques. A well managed SPL gets an advantage from the intercorrelation shared among all features [52]. It helps consider multiple goals such as tracking of requirements into products (i.e. better customization), software reuse development, evolutionary processes and software maintenance control [236].

Over the last decade, the substantial benefits of using the metaheuristics in SPL practices have been extensively argued in many research and practice. For example, Colanzi et al. [109] applied the NSGA-II approach which considered the modularity and extensibility as architectural features (objectives) to develop product line architecture. Segura et al. [237], and Lopez-Herrejon et al. [238] have suggested applying SBSE for the selection of the computational feature models for the SPL evaluations. Moreover, a fuzzy multi-objective approach (i.e. a combination of the fuzzy inference systems and NSGA-II) has been used by Cruz et al. [239] and Zhang et al. [240] to assist the decision makers in the management of the product lines. Other studies [241–245] used the genetic algorithm approach to build their models by searching for the related SPL design features to generate customer satisfaction.

2.5.3 Software Testing

Testing is a crucial stage in the software development life cycle. It represents around 50% of the total development cost [246]. Search-Based Software Testing (SBST) is a predominantly applied method to generate test cases automatically. In the soft-
2.5. Applications of the Search-Based Software Engineering

In software testing, metaheuristics are used to transfer testing cases into optimization problems in order to tackle hard problems [247]. SBST is thus a combination of the automatically generated test case and optimization search techniques that used for grabbing the testing cases (problems) in order to be solved [248].

The main goal of the SBST is to identify the errors while running software product either in a portion or the whole software product to ascertain a correct execution [249]. In software testing, a test case refers to a set of variables where a tester needs to satisfy the suitable requirements and working of software product under test. The main objective of the search based software testing process is test cases prioritization, test data generation, test suits minimization [248]. One of the most common and resource-intensive SBST methods is regression testing activity. Chittimalli et al. [250] argued that the regression testing process represents around 33% of the total software expenses. This process ensures that the software system progressions will not degrade the prior functioning software. In this context, under the test method, while the software system evolves, the system might experience some changes, i.e., the tendency of the software test suite increases in size hence increases the execution cost. Test suite minimization (in regression testing area [251]) is thus critical to decrease the retesting cost for the whole system [148]. The activity of the test case generation is to generate test cases to identify the errors and hence to result in a cost-effective and efficient testing technique. Test case generation is difficult as well as time-consuming process that completely relies on the tester [252].

A number of relevant search-based approaches have extensively explored test case generation in the SBST filed. For example, several experimental studies [253–256] have carried out in the context of test case generation to obtain structural test suites. These studies worked on adjusting (tuning) parameters of the adopted genetic algorithm approach. Other studies (e.g. [257–259]) were focused on object-oriented systems. In
2.5. Applications of the Search-Based Software Engineering

In this context, the authors applied different multi-objective approaches to address the problem of test case generation, or used for the comparative purpose.

Another frequently applied testing activity in the SBSE field is test case prioritization. It aims to generate a set of test cases to attain an early optimization based upon the preferred selected properties \(260\). The main goal of this activity is to improve the software test viability. Test case prioritization enables the adopted approach to conduct highly important test cases execution in order to provide desired outcomes such as earlier revealing for the faults and supplying feedback to the testers \(251\). For example, recent publications \(261, 262\) have supported the implementation of a multi-objective approach in test case prioritization as the approach is capable of solving two or more different objectives in one single prioritization. Thus, test case prioritization is considered as an important component that increases the effectiveness of the testing process in terms of time, cost, and the rate of fault detection.

Like test case generation, test case prioritization has also been widely investigated by several relevant search-based approaches. Khatibsyarbini et al. \(263\) conducted a systematic literature review regarding test case prioritization approaches. The review process showed that 25\% of the reviewed studies applied the search-based test case prioritization method. Another systematic literature review by Catal \(264\) argued that the performance of the genetic algorithm approaches is highly effective and can bring major benefits such as coverage and fault detection for test case prioritization. In practice, metaheuristic optimization algorithms have been used to tackle test case prioritization problem such as NSGA-II (e.g. \(265, 266\)), SPEA2 (e.g. \(267\)), Particle Swarm Optimization (e.g. \(268\)), GA (e.g. \(269, 272\)). The application of these approaches is aiming for enhancing the fault detection rate. Notice that, these approaches have been categorized regarding the domain and nature of the conducted study.

In SBSE field, both test case generation and prioritization tasks are highly im-
2.5. Applications of the Search-Based Software Engineering

2.5.4 Software Maintenance

In software engineering development tasks, it has been argued that the maintenance process represents about 70–75% of software development effort [273]. The search-based applications on software maintenance aim to preserve and improve the quality of the software system once it operates. According to Harman et al. [29], most of the existing software maintenance work focussed on refactoring and modularization.

The core function of the software modularisation process is to identify optimal modules or packages (i.e. the decomposition of a system into clusters of subsystems) [49]. It thus helps to understand the structure of the software system. Software modularisation is a search based problem that might adopt the same optimization procedure and metrics compared to the design task. In SBSE, The recurrently used metric is modularisation quality (MQ), which seeks for a trade-off between the low coupling and high cohesion to determine the classes need to be clustered in a package [274], aiming to improve the quality of this package (module). Note that, when these classes are grouped in a wrong package, it is difficult to understand the obtained design and hence unable to improve the system regarding cohesion and coupling.

Moreover, the suggestion of re-modularization solutions aims to enhance the structure of packages regardless of the number of code changes [273]. However, in a real-world framework, developers would rather have the re-modularized solutions to develop the structure of the system with the least number of changes. Minimizing the number of changes helps developers to understand the design better while applying the suggested changes. For example, Mkaouer et al. [276] used NSGA-III as a multi-objective search-based approach to finding the optimal re-modularization solutions that considered minimizing the number of changes, keeping the coherence of semantics, and
enhancing the structure of the identified packages. There has been extensive work on different software modularization’s tools and SBSE techniques (e.g. [277–283]). Mostly, these studies considered the clustering problem. They worked to find the best system decomposition in the regard of modules and not by enhancing the existing modularizations.

According to Fowler definitions [284] refactoring is “A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior”. The refactoring action is commonly used in the software maintenance process. It aims to modify the code structure without doing any change to the external functionality of the program (i.e software structure external behavior) [284]. When the maintenance refactorings are used in the software system, they can either the system quality or degrade it. Nevertheless, refactoring tools are applied to modify the original software solution and to improve quality design [285]. The main focus of the search-based software maintenance is to find optimal refactoring operations to prevent harming software (i.e. bad smells code), and hence to get a better and improved code quality.

In software maintenance, the SBSE algorithm can apply to refactorings to the code in order to decrease the technical uncertainty. As a baseline step, the developer applies this algorithm on the original program then moving from this step forward to improve the design quality of the software system [273]. A growing number of reviews have argued the concept of refactoring in search-based software engineering such as, refactoring to enhance software quality (e.g. [286–288]), refactoring as a test tool for software effort (e.g. [145, 289]), refactoring to evaluate metric effectiveness (e.g. [290, 291]), and refactoring for correcting software defects (e.g. [142, 292–294]).
2.5.5 Other Applications

There are other applications in different search-based software engineering areas. For example, in program comprehension, Fatiregun et al. [295, 296] proposed approaches for optimizing source code aiming to enhance the execution time by reducing code size. In co-evolutionary comprehension, Kessentini et al. [297] applied a multi-objective technique for the co-evolution process of initial models to generate a new model version of the better operation sequence aiming for the best compromise. In software correction, Wilkerson et al. [298] employed a combination of both NSGA-II with GP approaches to create test cases for detecting and fixing bugs automatically. In quality of service (QoS) evaluation for service composition, Ramirez et al. [299] employed two multi-objective approaches (NSGA-II and SPEA2) to optimize nine QoS properties. The main objective is to regain a better subset of properties (i.e. properties with better QoS values).

2.6 Chapter Summary

In this chapter, we have provided an overview of search based software engineering concepts, we have briefly described the metaheuristic search-based techniques including the well-known single and multi-objective evolutionary algorithms that have been used in this thesis: Genetic programming (GP), Genetic Algorithm (GA), Non-dominated Sorting Genetic Algorithm (NSGA-II), Random Search (RS), Indicator-Based Evolutionary Algorithm (IBEA), Multiobjective Cellular Genetic Algorithm (MOCell), S-metric Selection Evolutionary Multi-Objective Algorithm (SMS-EMOA), Strength Pareto Evolutionary Algorithm-II (SPEA2). We have also given a background of learning algorithms including Case-Based Reasoning (CBR), Random Forests (RF), Linear Regression (LR). We have then explained predictive performance metrics used in our
2.6. Chapter Summary

research. We followed that with a brief description of issue-driven in open software project including issue lifecycle and an iteration. In addition, we have explained software effort estimation, and modern code review. The final part of the chapter has broadly introduced the search-based software engineering analytics. Lastly, we have illustrated a number of application areas in search-based software engineering. In the next chapter, we commence describing our work which starts with iteration planning in agile development.
Chapter 3

Iteration Planning

Agile software development methods (e.g. Scrum) are widely used in the industry. The basis of agile methods is the incremental and iterative process in which software is developed through repeated cycles (iterative) and in smaller parts at a time (incremental). A project has a number of iterations (e.g. sprints in Scrum). An iteration is typically a short (usually 2–4 weeks) period in which the development team designs, implements, tests and delivers a distinct product increment, e.g. a working milestone version or a working release. Each iteration requires the completion of a number of user stories and/or tasks, which are usually recorded as issues in an issue tracking system (e.g. JIRA). This shifts the traditional model where all functionalities are delivered together (in a single delivery) to an agile, flexible model which involves a series of incremental deliveries and small iterations.

The inherent dynamic nature of software development (e.g. constant changes to software requirements) still introduces uncertainties regardless of which development process is employed. Hence, effective estimating and planning are still crucial for agile software development to cope with the need for rapid delivery. We thus focus on providing support for planning a single iteration at a time, rather than the whole software lifecycle as in traditional waterfall-like software development processes.
Substantial work in software analytics has been dedicated to build various prediction models to support software development. For example, most of existing work in effort estimation models (e.g. [14,301,302]) aim to predict the effort required for developing a whole software, rather than an iteration. Other work (e.g. [303–306]) focused on predicting at the project level. In modern agile settings, project managers and decision makers would, however, need insightful and actionable information at the level of iterations. Recent work using software analytics has started addressing this, such as estimating story points [307] or predicting that an iteration is at risk of not delivering what has been planned for [308].

It has now become a common practice for agile teams to plan for each iteration. An iteration plan provides a detailed picture that teams use to drive their work within an iteration. Central to an iteration plan is a set of issues which the team has decided to complete during the iteration. That means, iteration planning is an important activity in managing software projects where the software team needs to decide what should be done (in terms of issues) for an upcoming iteration. Selecting issues from the product backlog to form an iteration is however a challenging task as the selection needs to take into account a number of factors. For example, the combination of the selected issues needs to meet the goal that has been set out for the iteration. In addition, priorities and the effort of resolving issues and the team’s capability should be considered. The business value that a team delivers to the customers at the end of an iteration is also an important factor. Currently, most agile teams heavily rely on experts’ subjective assessment, and there is a serious lack of automated support which can help the team to arrive at an optimal selection of issues.

The work in this chapter aims to fill that gap. We employ a multi-objective search-based evolutionary approach to recommend the team select issues to be completed in an upcoming iteration. Specifically, we leverage a meta-heuristic technique, namely
3.1 Motivation Example

In modern agile development settings, a team maintains a product backlog which contains a list of things (e.g., user stories) that need to be done within the project. Those

genetic algorithms, to generate a large number of candidate selection of issues and search for the ones that are optimal with respect to a number of objectives. Our current work explores two objectives which guide our search algorithms. The first objective is to maximize the business value that an iteration delivers to the customers. In practice, each iteration has a goal which is defined by the team to state what will be accomplished during an iteration. Selected issues must therefore collectively achieve the iteration’s goal as much as possible. Hence, our second objective is to select issues that maximize this collective contribution towards the iteration’s goal. We name our approach Multi-Objective Search-Based Issue Iteration Planning (MOSBIP).

The evaluation demonstrates that our search-based approach outperforms the common baselines. We also demonstrate the effectiveness of using a multi-objective approach against the single objective approach. The evaluation was performed against a dataset of 233 iterations (which consist of 55662 issues in total) we collected from six different Apache projects. Following common standards, we use precision, recall and F-measure to evaluate the performance of our models, and also use a non-parametric Wilcoxon test \( [309] \) and Vargha and Delaney’s statistic \( [310] \) to demonstrate both the statistical significance and the effect size of the results.

The remainder of this chapter is organized as follows. Section 3.1 formulates the problem definition of iteration planning. Section 3.2 describes our multi-objective approach to solve this problem using evolutionary algorithms. Section 3.3 reports on the experimental evaluation of our approach. Related work is discussed in Section 3.4 before we conclude and outline future work in Section 3.5.
things (e.g. implementing new feature, making improvements, and/or fixing bugs) are recorded as *issues* if the team uses an issue-tracking system (e.g. JIRA). Figure 3.1 shows an example of a product backlog of project FeedHenry in the JBoss repository. The backlog had 120 issues, all of which had not been resolved.

![Figure 3.1: Example of a product backlog in JBoss](image)

Issues in the backlog are prioritized, which reflects the urgency and importance of the tasks from the client’s (e.g. product owner) perspective. An issue has a number of attributes such as type, priority, story points, and a textual description (see Figure
Motivation Example

The type of an issue specifies the nature of an issue (e.g. new feature, defect) where the textual description (e.g. summary or description) provides the detailed nature of the issue. The priority of an issue reflects its importance from the client’s perspective (e.g. new functionality urgently needed or a critical bug which must be fixed as soon as possible). The story points represent an estimate of the size of an issue. Thus, it reflects how the business values of different issues are relative to each other. For example, an issue which has 4 story points bring twice as much value as the one with 2 story points.

![Example of an issue](https://issues.jboss.org/browse/FH-2649)

**Figure 3.2:** Example of an issue

---

1https://issues.jboss.org/browse/FH-2649
3.1. Motivation Example

Agile projects are organised in terms of time-boxed iterations (alternatively called *sprints*). In each iteration/sprint, the team commits to deliver a certain amount of work (e.g. implementing new feature, making improvements, and/or fixing bugs). As part of planning for an upcoming iteration/sprint, the team needs to define the *iteration/sprint goal*, which is an objective set for the iteration/sprint that can be achieved through completing a subset of issues in the product backlog. The iteration goal is usually a short text description. For example, Figure 3.3 shows that the iteration *Thai Sprint 3* in JBoss has the goal “70% project done: from an EC2 instance having an nginx process running, a user can get a new instance with nginx container created”.

A critical part of iteration planning is that the team needs to decide which issues in the product backlog will be selected for resolving in the upcoming iteration. There are a number of factors which the teams need to consider here. Firstly, the selected issues must collectively achieve the iteration’s goal, i.e. they need to align with the goal. For example, all the issues selected for the iteration *Thai Sprint 3* in JBoss contribute to accomplishing this iteration’s goal, e.g. list EC2 instances that are not imported or cloned, cloned the prepared instance, etc. (see Figure 3.3). Secondly, since the team aims to deliver as much immediate business values to the customers as early as possible, issues that have higher priority and larger story points tend to be selected first. For example, issue *FH-2699* is near the bottom of the prioritized backlog, and has only 1 story point, and thus was not included in the upcoming iteration *Thai Sprint 3*.

Thirdly, it is also important to take the team’s capability into consideration. If many issues are selected, the team may not be able to complete them within an iteration. On the other hand, if there are only a few issues, the team may complete

[https://issues.jboss.org/secure/RapidBoard.jspa?rapidView=3172&projectKey=FH&view=reporting&chart=sprintRetrospective&sprint=7211](https://issues.jboss.org/secure/RapidBoard.jspa?rapidView=3172&projectKey=FH&view=reporting&chart=sprintRetrospective&sprint=7211)
3.1. Motivation Example

them well before the iteration ends. In practice, agile teams often use velocity, a simple but powerful method for measuring the rate at which the teams consistently deliver business value in each iteration. The velocity reflect the team’s capability in delivering business values, and is calculated by summing up the story points of all the issues which the team successfully delivered in an iteration. For example, Thai Sprint 3 delivered 43 story points.

Considering all of those factors when selecting issues for an upcoming iteration is thus a challenging task for agile teams. Product backlog can be large and thus there

Figure 3.3: Example of a sprint report of sprint Thai Sprint 3
are many potential selections to be considered. In the above example, there are 120
issues in the product backlog, and thus there are $120!$ combinations to be considered.
Agile teams currently do not have much automated support when facing this large
scale of complexity. Our work aims to fill this gap. We formulate this as a multi-
objective optimization problem, and employ an evolutionary approach to develop a
machinery which supports the team in selecting issues for an upcoming iteration.

3.2 Approach

We leverage a search-based software engineering (SBSE) approach to support the team
in selecting issues from the product backlog for an upcoming iteration. Our approach
employs evolutionary techniques to iteratively generate candidate selections of issues
for a given iteration. Each selection represents a candidate solution in the search space.
The search is guided simultaneously by two objectives: maximizing the business value
which the team delivers in the iteration, and maximizing the alignment of the selected
issues with the iteration’s original goal. The evolutionary algorithms we employed (e.g.
NSGA-II) works based on the principle that a population of candidate solutions to an
optimization problem is evolved toward better solutions, following Darwin’s evolution
theory. Each candidate solution has a number of properties (i.e. chromosomes or
genotype) which can be mutated and altered to derive new candidate solutions. At
the end of the search process, the machinery returns a set of non-dominated solutions,
each of which represents a subset of issues in the product backlog that the team can
select for the upcoming iteration. We now describe our approach in details.

3.2.1 Solution Representation

We represent each candidate solution (i.e. a set of issues in the product backlog) using a
bit string which has the number of bits equal to the size of the product backlog. For ex-
ample, if there are seven issues (i.e. AURORA-1014, AURORA-1225, AURORA-1681, AURORA-1767, AURORA-1771, AURORA-1777, and AURORA-1688) in the product backlog, we can represent a solution as a bit string with seven bits (see Figure 3.4). Each bit in the string corresponds to an issue, and has the value of either 0 or 1. A bit is set to 1 if the corresponding issue is selected for the upcoming iteration, and 0 if the issue is excluded. For example, the bit string 0110111 indicates that AURORA-1225, AURORA-1681, AURORA-1771, AURORA-1777, and AURORA-1688 are selected for the upcoming iteration while AURORA-1014 and AURORA-1767 are excluded.

<table>
<thead>
<tr>
<th>Set of issues in the backlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
</tr>
<tr>
<td>AURORA-1014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subset of predicted issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obtained predicted issues after the &quot;0&quot; value has been rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_2$</td>
</tr>
<tr>
<td>AURORA-1225</td>
</tr>
</tbody>
</table>

Figure 3.4: An example of how a candidate solution is represented using a bit string.

### 3.2.2 Fitness functions

The search for the best solutions is guided by a number of fitness functions, each of which reflects the “quality” of a solution with respect to a certain criterion. Our approach employs two fitness functions: maximize the business value which the team can deliver to the customers at the end of an iteration, and maximize the degree of alignment between the selected issues and the iteration’s goal. The details of these fitness functions are described as below.
3.2.2.1 Maximize Business Value

Delivering real business value to the customers is the most important progress measure for an agile development team. Business values can be in various forms, such as adding new functionalities to a software product that significantly create business advantages for the customers, or fixing some critical bugs reported by the customers, or improving the performance of the existing functionality. The business value which a team delivers to the customer in an iteration resides in the issues that the team has completed at the end of the iteration. This business value can be quantified in terms of two dimensions. First, the importance of an issue reflects what the customers really need at a particular time during the project. Agile teams regularly prioritize issues in the backlog (i.e. assigning priority level to the issues) as a way to increase the level of customers satisfaction by continuous delivering what the customers really need. Second, the business value of an issue is also reflected through the size and complexity of the issue, e.g. larger functionalities delivering higher business value to customers. The size and complexity of an issue is commonly measured in terms of story points. Story points are relative values, i.e. an issue that is assigned four story points deliver twice as much value as an issue assigned two story points.

Hence, the first objective is to maximize the business value which the team delivers at the end of an iteration. We defined this business value as a function over the priority and story points of the issues selected for the iteration. This fitness function is defined as follows.

\[
\text{BusinessValue}(S) = \sum_{i \in S} SP(i) \times \text{Priority}(i) \tag{3.1}
\]

where \( S \) is a set of selected issues, \( SP(i) \) is the size of issue \( i \) in \( S \) (measured in terms of story points), and \( \text{Priority}(i) \) is the priority level of issue \( i \).
The issue priority can be translated to a numerical scale. For example, we used ordinal values from 1 to 5 where 1 represents the least importance (e.g. trivial issues) and 5 represents the most importance (e.g. blocker issues).

### 3.2.2.2 Maximize Goal Alignment

The product owner and the team need to select issues that combine to achieve the iteration goal. Since issues and iteration goals are described in natural language text, we use traditional textual similarity metrics to measure the degree of alignment between a set of issues and an iteration goal. First, we compute the cosine similarity between the iteration goal $G$ and issue $i$ as below.

$$ Similarity(G, i) = \frac{\vec{V}_G \cdot \vec{V}_i}{|\vec{V}_G||\vec{V}_i|} \quad (3.2) $$

where $\vec{V}_G$ and $\vec{V}_i$ are a vector of term weights for the textual description of $G$ and $i$ respectively.

The term weights are calculated using the *term frequency* ($tf$) and the *inverse document frequency* ($idf$), which are defined as follows:

$$ tf(t, desc) = \frac{f_{t,desc}}{T} \quad \text{and} \quad idf(t) = \log \left( \frac{D}{n_t} \right) \quad (3.3) $$

where $f_{t,desc}$ is the number of time term $t$ occurs in description $desc$, $T$ represents the total number of terms in description $desc$, $n_t$ is the number of descriptions that contain the term $t$, and $D$ represents the total number of documents in the corpus.

Each term weight $w$ in vector $\vec{V}_G$ is computed as $w_{t \in G} = tf(t, G) \times idf(t)$. Thus,
\( \vec{V}_G \cdot \vec{V}_i \) represents the inner product of the two vectors and is computed as follows:

\[
\vec{V}_G \cdot \vec{V}_i = \sum_{t \in (G \cup i)} (tf(t, G) \times idf(t)) \times (tf(t, j) \times idf(t)) \quad (3.4)
\]

and \( |\vec{V}_G| \) and \( |\vec{V}_i| \) are calculated as follow:

\[
|\vec{V}_G| = \sqrt{\sum_{t \in G} (tf(t, G) \times idf(t))^2} \quad (3.5)
\]
\[
|\vec{V}_i| = \sqrt{\sum_{t \in i} (tf(t, i) \times idf(t))^2} \quad (3.6)
\]

Let \( \{i_1, i_2, ..., i_p\} \) be the (sub)set of issues selected from the product backlog (denoted as \( S \)). The degree of alignment of a set of issues \( S \) with the iteration goal \( G \) is defined as follows.

\[
\text{GoalAlignment}(G, S) = \sum_{i \in S} \text{Similarity}(G, i) \quad (3.7)
\]

Issues are selected such that the above goal alignment is maximized. Thus, this goal alignment function is used as the second fitness function in our approach.

### 3.2.3 Constraints

The selection of issues for an upcoming iteration is constrained by the team’s capability. We use the well-known velocity measure to represent the team’s capability in delivering business value. The velocity is calculated by adding up the story points of all the issues which the team successfully delivered in an iteration:

\[
Velocity(S) = \sum_{i \in S} SP(i) \quad (3.8)
\]
Where $S$ is the set of issues completed in an iteration, and $SP(i)$ is the size of issue $i$ in terms of story points.

We calculate the range of values which is likely to contain the team’s velocity for the upcoming iteration $k + 1$, and use this to constrain the selection of issues. To do so, we construct the confidence intervals at a selected confidence level (e.g. 95%) using the team’s velocity from the previously completed $k$ iterations. The confidence intervals are computed as follows.

\[
UpperLimit = \bar{x} + Z_{a/2} \times \frac{\sigma}{\sqrt{k}}
\]

\[
LowerLimit = \bar{x} - Z_{a/2} \times \frac{\sigma}{\sqrt{k}}
\]  \hspace{1cm} (3.9)

where $\bar{x}$ is the mean velocity of the previously completed $k$ iterations, $(Z_{a/2})$ is the confidence coefficient (computed from the cumulative distribution function of normal distribution) where $a$ is the confidence level (e.g. $Z_{a/2} = 1.96$ for 95% confidence level), $\sigma$ is the standard deviation. Any feasible solution $S$ must have the velocity falling into this range, i.e. $Velocity(S) \in [UpperLimit, LowerLimit]$.

### 3.2.4 Evolutionary Search

The search for a subset of issues in the product backlog to be included in the upcoming iteration starts with an initial population in which each individual in the population is a candidate set of issues. The initial population is created by randomly generating a number of sets of issues. The fitness values of each individual with respect to each of two fitness functions (see Section 5.2.5) are computed. The population is then undergone a selection process.

Selected individuals form the parent to generate a new generation of individuals through the crossover and mutation operators. These genetic operators act directly on the representation of candidate solutions (i.e. bit strings – refer to Section 5.2.3).
to form new valid representations. The mutation operator randomly chooses certain
bits in the string and set them to an opposite value (i.e. inverted from 0 to 1 and
vice versa). The crossover operators involve two parent bit strings (representing two
candidate solutions). A crossover point on both parents’ strings is chosen. The parts
beyond that point in both parents’ strings are then swapped to generate the offspring
strings. This evolution process continues until a fixed number of generations has been
reached.

We search for solutions that meet both objectives: maximize business value de-

erived to the customers and maximize the alignment to the iteration goal. These
solutions form a Pareto front of business value and goal alignment. A solution on a
Pareto front does not dominate another solution on the same front, i.e. the former
is better than the latter with respect to at least one objective (e.g. business value),
and not worse in the other objective (e.g. goal alignment). For example, in Figure
$S_1$ solution does not dominate $S_2$ solution since the former is lower than the lat-
ter in terms of the goal alignment objective but has greater business value. On the
other hand, $S_1$ dominates $S_4$ since the $S_1$ has a lower goal alignment than $S_4$ and also
has smaller business value. A range of multi-objective optimization algorithms can be
used to find a Pareto front. We have investigated a number of them in our evaluation
(refer to Section 3.3). One of the most widely-used algorithms is the non-dominated
sorting genetic algorithm (NSGA-II). The NSGA-II is discussed in more details in the
background chapter, Section 2.1.3.1.

### 3.2.5 Selecting a Solution From a Pareto Front

Our machinery returns a Pareto front of solutions. Choosing which one of these
solutions to use is often a user-specific decision. Different approaches have also been
proposed to help select a single solution from a Pareto front (e.g. knee points 312).
3.2. Approach

In particular, the knee point approach have widely been used in previous work (e.g. [313–316]). This approach measures the Euclidean distance of each solution on the Pareto front from the reference point. This reference point has the optimal values for each objective function. The solution we select (denoting as \(ROS\)) is the one closest to the reference point (i.e. minimizing the distance). Formally, given a Pareto front \(P = (S_1, S_2, S_3, \ldots, S_P)\) where \((S\) solutions including knee point \(S_K) \in P\), and assume that the reference point has the maximum business value \(BV_{\text{max}}\) and goal alignment \(GoA_{\text{max}}\), we calculate the \(ROS\) using the following formula:

\[
ROS = \sqrt{(BV_{\text{max}} - BV(S_i))^2 + (GoA_{\text{max}} - GoA(S_i))^2}
\]  

(3.10)

where \(S_i \in P\). In our evaluation, we selected a solution from the Pareto fronts returned by our machinery using this knee point technique.

Figure 3.5: An example of non-dominated fronts
3.3 Evaluation

This section discusses the evaluation that we have carried out for our approach. We first describe how data is collected and preprocessed for our study. Next, we describe the experimental settings, discuss the performance measures, and report our results.

Our empirical evaluation aims to answer the following research questions:

- **RQ1. Sanity Check:** *Is the multi-objective search-based approach suitable for selecting issues for an upcoming iteration?*

  This sanity check requires us to compare our multi-objective approach MOSBIP against the baseline technique namely random guessing. It is a naive technique [14, 208] which randomly choose issues from the product backlog. The entire process is repeated 1000 times, and we then take the mean performance.

- **RQ2. Different multi-objective optimization algorithms:** *Which multi-objective optimization algorithms perform best with our approach?*

  Our approach is generic in which different multi-objective optimization algorithms can be used. Although NSGA-II is the main algorithm (described in details in Section 5.2.2 that we employed, there are also other suitable algorithms for our approach. We have tested our approach with Random Search (a common naive benchmark), and two recently developed multi-objective evolutionary algorithms: Multiobjective Cellular Genetic Algorithm (MOCell) [161] and the Strength-based Evolutionary Algorithm (SPEA2) [317].

- **RQ3. Benefits from Multi-objective Approach:** *Does our multi-objective approach provide more accurate and robust recommendation than alternative single-objective approaches?* To answer this question, we implemented the traditional single-objective genetic algorithms using Business Value (BV) or the goal alignment (GoA) as the objective function. We name these alternative ap-
proaches as \textit{GA-BV} and \textit{GA-GoA}. We then compare the performance of our proposed approach (MOSBIP) against these two single-objective approaches.

### 3.3.1 Datasets

In this section, we describe how data were collected for our empirical study and the experiments, and we also discussed about the dataset used in this work that includes the data pre-processing stage and the statistical information of this dataset. Our datasets for iteration planning is the first dataset of its type.

#### 3.3.1.1 Data Collecting

We collected data of past sprints and issues from the Apache repository. Originally, Apache is a web server, but currently the Apache repository hosts more than fifty sub-projects under their community (e.g. AURORA, MAHOUT). All issues and sprints in Apache are recorded in Apache’s issue tracking system\(^3\). We then collected 6 projects from Apache namely: MESOS, USERGRID, AURORA, SLIDER, KYLIN and MAHOUT. These projects follow the Agile practices for their development and use \textit{JIRA-Agile}\(^4\) to support their sprints and issues management. We used the Representational State Transfer (REST) API provided by JIRA-Agile for querying the issue reports and sprint reports JavaScript Object Notation (JSON) format. Note that JIRA-Agile supports both the Scrum and Kanban practices. We however collected only the sprint reports following the Scrum practice.

Initially, we collected 589 sprints from six projects and 133,215 issues involved with those sprints from July 10, 2013 (the date when the first iteration was created) to November 23, 2017 (the date when we finished collecting the data).  

\(^3\)https://issues.apache.org/jira  
\(^4\)JIRA-Agile is a well-known issue and project tracking tool that supports agile practices.
### Table 3.1: Descriptive statistics of the sprints of the projects in our datasets

<table>
<thead>
<tr>
<th>Project</th>
<th>#issues</th>
<th>#sprints</th>
<th># issues/sprint</th>
<th>#days/sprint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min/Max</td>
<td>Mean</td>
<td>Median</td>
<td>SD</td>
</tr>
<tr>
<td></td>
<td>Min/Max</td>
<td>Mean</td>
<td>Median</td>
<td>SD</td>
</tr>
<tr>
<td>MESOS</td>
<td>40649</td>
<td>125</td>
<td>116/886</td>
<td>456.74</td>
</tr>
<tr>
<td></td>
<td>427</td>
<td>189.87</td>
<td>5/18</td>
<td>13.46</td>
</tr>
<tr>
<td>USERGRID</td>
<td>6705</td>
<td>53</td>
<td>14/344</td>
<td>248.34</td>
</tr>
<tr>
<td></td>
<td>247</td>
<td>69.08</td>
<td>6/14</td>
<td>10.88</td>
</tr>
<tr>
<td>AURORA</td>
<td>5721</td>
<td>27</td>
<td>7/258</td>
<td>211.89</td>
</tr>
<tr>
<td></td>
<td>222</td>
<td>52.53</td>
<td>4/14</td>
<td>11.63</td>
</tr>
<tr>
<td>SLIDER</td>
<td>901</td>
<td>13</td>
<td>19/114</td>
<td>69.31</td>
</tr>
<tr>
<td></td>
<td>63</td>
<td>30.34</td>
<td>9/14</td>
<td>13.19</td>
</tr>
<tr>
<td>KYLIN</td>
<td>1244</td>
<td>8</td>
<td>29/198</td>
<td>155.5</td>
</tr>
<tr>
<td></td>
<td>171</td>
<td>52.98</td>
<td>10/14</td>
<td>13</td>
</tr>
<tr>
<td>MAHOUT</td>
<td>374</td>
<td>6</td>
<td>76/166</td>
<td>115.67</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>45.94</td>
<td>26/47</td>
<td>34.34</td>
</tr>
<tr>
<td>Total</td>
<td>55662</td>
<td>233</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 3.3.1.2 Data Pre-processing

We performed the pre-processing step to build the datasets for our experiments. Our approach needs two sets of information to make a recommendation: a backlog – i.e. list of on-going issues with their attributes (story point, priority, and textual description), and a list of actual completed issues from a sprint – i.e. ground truth. A sprint report provides the list of completed, in-completed, and removed issues which we can use the completed issues as our ground-truth. We acknowledge that the information of in-completed and removed issues could provide benefits. However, we leave it for the future work since this work focuses on studying the objectives for recommending potential issues completed from a sprint.

To mimic the scenario, we collected all issue from each project (e.g. more than 10,000 issue reports were collected from the MESOS project). We thus built a backlog for each sprint by investigating issue’s change logs where a backlog for each sprint contains only the issues having their creation date before the sprint start date and having their resolved date after the sprint start date. We then used the sprint start date as a reference point to extract issue’s attributes. For example, an issue has been assigned to low priority after resolving, but it is a high priority at a sprint start date. We removed sprints contained zero completed issues and not having close status. There are some issues have not been assigned a story point. We filled those missing story points with 1.
In total, we conducted our study on 233 sprints from six projects, which include 55,662 issues. Table 3.1 summarizes the descriptive statistics for the sprints and issues of our dataset in terms of number of issues per backlog and sprint, the sprint length, minimum, maximum, mean, median, and standard deviation. Across all the six projects, the number of issues per sprint varies and the sprint length tends to be in the range of 2 to 4 weeks. For example, the mean number of issues per sprint in MESOS is 456.74 issues, while it is only 69.31 issues in SLIDER. JIRA allows the users to explicitly define the goal of a sprint since version 7.5. Older versions do not support this feature. Many of the sprints that we collected were created in the older versions, and they do not have a sprint goal explicitly defined. Hence, we needed to reconstruct the goal of those sprints using the issues that was delivered in the sprints. Specifically, we used Latent Dirichlet Allocation (LDA) to build a topic model over the descriptions of all the issues in each sprint, and used the top 20 topics to represent the goal of that sprint. Note that this was purely for reconstructing sprint goals. Our search algorithms did not use any of these “future” information to inform the search.

3.3.2 Experimental Settings and Measures

This section describes the experimental setup and the performance evaluation used for the results analysis. In this work, each project has a number of sprints (233 sprints in total). We ran each algorithm 30 times for each sprint, calculated the performance, and took the mean result.

To evaluate the performance of our proposed method, we employed precision, recall, and F-measure in Information Retrieval (IR) and commonly used for evaluating recommendation system. Note that we measure the performance of the model for each iteration in a project, and then report the average performance across
3.3. Evaluation

all iterations in the project. The performance measures can be defined as:

**Precision (Prec):** The ratio of the correctly recommended issues over all the recommended issues. It is calculated as:

\[
P_{\text{prec}} = \frac{|\text{Actual}_i \cap \text{Recommended}_i|}{|\text{Recommended}_i|} \tag{3.11}
\]

\[
\text{Avg(Prec)} = \frac{1}{m} \sum_{i=1}^{m} P_{\text{prec}_i}
\]

**Recall (Re):** The ratio of the correctly recommended issues over all the actual completed issues. It is calculated as:

\[
R_{\text{e}} = \frac{|\text{Actual}_i \cap \text{Recommended}_i|}{|\text{Actual}_i|} \tag{3.12}
\]

\[
\text{Avg(Re)} = \frac{1}{m} \sum_{i=1}^{m} R_{\text{e}_i}
\]

**F-measure (F1):** The weighted harmonic mean of the precision and recall. It is calculated as:

\[
F_{1} = \frac{2 \cdot (\text{Prec}_i \cdot \text{Re}_i)}{\text{Prec}_i + \text{Re}_i} \tag{3.13}
\]

\[
\text{Avg}(F1) = \frac{1}{m} \sum_{i=1}^{m} F_{1_i}
\]

Where \( m \) is the number of sprints in a project, \text{Recommended}_i is a set of issues recommended by an approach for iteration \( i \), and \text{Actual}_i is a set of issues actually delivered in this sprint \( i \).

In order to compare the performance of two models, we employ the Wilcoxon Signed Rank Test [309] to assess the statistical significance of the precision, recall, and F-measure achieved with the two models. The Wilcoxon Signed Rank Test does not assume a normal distribution in the data which is a safe test. The null hypothesis here is: “the performance provided by an our approach are not different to those provided by alternative approaches”, which we work to reject this null hypothesis. We set the
confidence limit at 0.05 (i.e. $p < 0.05$). We then assessed whether the effect size is interesting by employing the correlated samples case of the Vargha and Delaney’s $\hat{A}_{XY}$ non-parametric effect size measure [310]. The $\hat{A}_{XY}$ measures the probability that the performance achieved from model X is better than the performance achieved from model Y. Note that we have 3 performance measures: precision, recall, and F-measure. We thus employ the statistical testing and effect size testing on each individual measure (i.e. precision, recall, and F-measure) which can be defined as the following formula (let take recall as an example):

$$\hat{A}_{XY}(Re) = \frac{\#(X_{Re} > Y_{Re}) + (0.5 \times \#(X_{Re} = Y_{Re}))}{m}$$

(3.14)

where $\#(X_{Re} > Y_{Re})$ is the number of sprints that the recall (i.e. $Re_{i}$) from model X more than the recall from model Y, $\#(X_{Re} = Y_{Re})$ is the number of sprints that the recall from model X equal to the recall from model Y, and $m$ is the number of sprints. We then calculated $\hat{A}_{XY}(Prec)$ and $\hat{A}_{XY}(F1)$ from the same formula.

We also use hypervolume [177] as a quality indicator for the volume of the space covered by the non-dominated solutions. This measure has been used in previous work (e.g. [14,324,325]) to as a performance indicator for multi-objective optimization. It reflects the convergence and diversity of the solutions on a Pareto front (e.g. the higher hypervolume, the better performance).

Our approach was implemented in the MOEA Framework[5]. We used the parameters that have been commonly used in previous search-based software engineering work [14,326]. Specifically, we employed tournament selection method and set the size of the initial population to 100. The number of generations was set to 100,000. Crossover probability was set to 0.9, mutation probability was 0.1, and reproduction probability was 0.2.

[http://moeaframework.org/index.html]
3.3.3 Results

In this section, we report the evaluation results of our approach to answer our research questions.

Results for RQ1:

We compare the performance achieved from our approach MOSBIP against the random guessing method. Table 3.2 shows the results achieved from MOSBIP and Random Guessing (RG). The analysis of all measures (i.e. precision, recall, and F-measure) suggests that the recommendation obtained with our approach, MOSBIP, are better than those achieved by using RG. MOSBIP consistently outperforms RG in all 6 cases. Our approach improved between 319.04% (in AURORA) to 380.95% (in MAHOT) in terms of precision, 433.33% (in KYLIN) to 718.18% (in AURORA) in terms of recall, and 394.11% (in KYLIN) to 550% (in MAHOUT) in terms of F-measure over the random guessing method.

Table 3.6 shows the results of the Wilcoxon test and the corresponding $\hat{A}_{XY}$ effect size to measure the statistical significance and effect size of the improved accuracy achieved by our approach over the random guessing. Our approach significantly outperforms the random guessing ($p < 0.001$) with effect sizes greater than 0.83 which can be considered as large effect size ($\hat{A}_{XY} > 0.8$), in all cases and for all three measures.

Our proposed approach, MOSBIP, outperforms the random guessing in all six open source projects, thus passing the sanity check required by RQ1.

Results for RQ2:

Table 3.3 shows the results from using different multi-objective optimization algorithms: Random Search, MOCell, and SPEA2. In terms of Random Search, our

---

6All the experiments were run on a Microsoft Windows 10 Home PC with an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz and 8.00 GB RAM.
approach using NSGA-II improved between 117.54% (in MESOS) - 321.73% (in AURORA) precision, 152.00% (in SLIDER) - 213.51% (in AURORA) recall, 137.72% (in MESOS) - 288.46% (in USERGRID) F-measure, and 137.89% (in SLIDER) - 275% (in MAHOUT) hypervolume.

NSGA-II improved over MOCell between 117.54% (in AURORA) - 135.59% (in MAHOUT) precision, 117.46% (in MAHOUT) - 154.90% (in MESOS) recall, 120.33% (in KYLIN) - 135.18% (in MESOS) F-measure, and 127.12% (in MESOS) - 136.54% (in KYLIN) hypervolume.

In terms of SPEA2, NSGA-II improved between 111.66% (in MESOS) - 145.65% (in AURORA) precision, 116.07% (in KYLIN) - 164.58% (in AURORA) recall, 123.72% (in MESOS) - 165.90% (in AURORA) F-measure, and 123.64% (in SLIDER) - 147.06% (in USERGRID) hypervolume.

The Wilcoxon test (see Table 3.4) also confirms that the improvement of our approach is significant ($p < 0.001$) with effect sizes greater than 0.6 in all cases.

Our proposed approach using NSGA-II significantly outperforms the three other alternative algorithms: the random search, MOCell, and SPEA2.

Results for RQ3:

Table 3.2 also shows the results from using single-objective genetic algorithm: GA-GoA and GA-BV. MOSBIP using multi-objective genetic algorithms improved between 124.07% (in AURORA) - 163.26% (in MAHOUT) in terms of precision, 100.00% (in KYLIN) - 168.08% (in AURORA) in terms of recall, and 128.84% (in KYLIN) - 148.97% (in AURORA) in terms of F-measure over the model using single-objective genetic algorithm: GA-GoA and GA-BV.

The Wilcoxon test also show the comparison between our Multi-objective MOSBIP and single-objective (GA-BV & GA-GoA) approaches. In Table 3.3 it can be
seen that, the improvement of the multi-objective approach MOSBIP over the single-objective is significant ($p < 0.001$) in 12/36 cases with the effect size greater than 0.62 all cases.

Using multi-objective approach provides more accurate and robust recommendation than single-objective models.

**Table 3.2:** Evaluation results of MOSBIP (our approach using NSGA-II), Random Guessing (RG), the single objective approaches GA-GoA and GA-BV

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLIDER</td>
<td>MOSBIP</td>
<td>0.72</td>
<td>0.76</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>RG</td>
<td>0.21</td>
<td>0.11</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.49</td>
<td>0.61</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>GA-BV</td>
<td>0.52</td>
<td>0.60</td>
<td>0.55</td>
</tr>
<tr>
<td>MAHOUT</td>
<td>MOSBIP</td>
<td>0.79</td>
<td>0.74</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>RG</td>
<td>0.21</td>
<td>0.11</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.49</td>
<td>0.66</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>GA-BV</td>
<td>0.53</td>
<td>0.59</td>
<td>0.56</td>
</tr>
<tr>
<td>AURORA</td>
<td>MOSBIP</td>
<td>0.67</td>
<td>0.79</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>RG</td>
<td>0.22</td>
<td>0.16</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.52</td>
<td>0.60</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>GA-BV</td>
<td>0.54</td>
<td>0.47</td>
<td>0.49</td>
</tr>
<tr>
<td>KYLIN</td>
<td>MOSBIP</td>
<td>0.74</td>
<td>0.65</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>RG</td>
<td>0.20</td>
<td>0.15</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.53</td>
<td>0.65</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>GA-BV</td>
<td>0.54</td>
<td>0.49</td>
<td>0.51</td>
</tr>
<tr>
<td>USERGRID</td>
<td>MOSBIP</td>
<td>0.70</td>
<td>0.80</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>RG</td>
<td>0.21</td>
<td>0.11</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.51</td>
<td>0.60</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>GA-BV</td>
<td>0.54</td>
<td>0.53</td>
<td>0.51</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOSBIP</td>
<td>0.67</td>
<td>0.79</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>RG</td>
<td>0.18</td>
<td>0.12</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.48</td>
<td>0.64</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>GA-BV</td>
<td>0.49</td>
<td>0.56</td>
<td>0.51</td>
</tr>
</tbody>
</table>
### Table 3.3: Evaluation results for MOSBIP using different multi-objective optimization algorithms (Random Search, MOCell, and SPEA2)

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>Hypervolume</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AURORA</strong></td>
<td>NSGA-II</td>
<td>0.67</td>
<td>0.79</td>
<td>0.73</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>Random Search</td>
<td>0.28</td>
<td>0.37</td>
<td>0.31</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.57</td>
<td>0.62</td>
<td>0.59</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.46</td>
<td>0.48</td>
<td>0.44</td>
<td>0.51</td>
</tr>
<tr>
<td><strong>MAHOUT</strong></td>
<td>NSGA-II</td>
<td>0.79</td>
<td>0.74</td>
<td>0.77</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>Random Search</td>
<td>0.41</td>
<td>0.42</td>
<td>0.41</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.59</td>
<td>0.63</td>
<td>0.61</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.60</td>
<td>0.62</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td><strong>SLIDER</strong></td>
<td>NSGA-II</td>
<td>0.72</td>
<td>0.76</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>Random Search</td>
<td>0.33</td>
<td>0.50</td>
<td>0.40</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.57</td>
<td>0.60</td>
<td>0.59</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.59</td>
<td>0.55</td>
<td>0.56</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td><strong>NSGA-II</strong></td>
<td>0.74</td>
<td>0.65</td>
<td>0.67</td>
<td>0.77</td>
</tr>
<tr>
<td><strong>KYLIN</strong></td>
<td>Random Search</td>
<td>0.23</td>
<td>0.33</td>
<td>0.26</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.58</td>
<td>0.52</td>
<td>0.55</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.52</td>
<td>0.56</td>
<td>0.54</td>
<td>0.56</td>
</tr>
<tr>
<td><strong>USERGRID</strong></td>
<td>NSGA-II</td>
<td>0.70</td>
<td>0.80</td>
<td>0.75</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>Random Search</td>
<td>0.30</td>
<td>0.42</td>
<td>0.26</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.55</td>
<td>0.58</td>
<td>0.58</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.52</td>
<td>0.54</td>
<td>0.53</td>
<td>0.64</td>
</tr>
<tr>
<td><strong>MESOS</strong></td>
<td>NSGA-II</td>
<td>0.67</td>
<td>0.79</td>
<td>0.73</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>Random Search</td>
<td>0.37</td>
<td>0.49</td>
<td>0.42</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>MOCell</td>
<td>0.57</td>
<td>0.51</td>
<td>0.54</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>0.60</td>
<td>0.61</td>
<td>0.59</td>
<td>0.64</td>
</tr>
</tbody>
</table>
### Table 3.4:  Comparison of MOSBIP vs. RS, MOCell, and SPEA2 using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets).

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>Hypervolume</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLIDER</td>
<td>NSGA-II vs RS</td>
<td>&lt;0.001 [0.93]</td>
<td>&lt;0.001 [0.93]</td>
<td>&lt;0.001 [0.92]</td>
<td>&lt;0.001 [0.92]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs MOCell</td>
<td>0.017 [0.77]</td>
<td>0.200 [0.70]</td>
<td>0.004 [0.85]</td>
<td>&lt;0.001 [0.92]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs SPEA2</td>
<td>0.002 [0.77]</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.84]</td>
<td>&lt;0.001 [0.85]</td>
</tr>
<tr>
<td>MAHOUT</td>
<td>NSGA-II vs RS</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.87]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs MOCell</td>
<td>0.109 [0.69]</td>
<td>0.019 [0.83]</td>
<td>0.007 [0.84]</td>
<td>0.007 [0.83]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs SPEA2</td>
<td>0.011 [0.66]</td>
<td>0.039 [0.83]</td>
<td>0.078 [0.83]</td>
<td>&lt;0.001 [0.83]</td>
</tr>
<tr>
<td>AURORA</td>
<td>NSGA-II vs RS</td>
<td>&lt;0.001 [0.96]</td>
<td>&lt;0.001 [0.94]</td>
<td>&lt;0.001 [0.96]</td>
<td>&lt;0.001 [0.96]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs MOCell</td>
<td>0.010 [0.77]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.94]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs SPEA2</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.92]</td>
<td>&lt;0.001 [0.87]</td>
<td>&lt;0.001 [0.94]</td>
</tr>
<tr>
<td>KYLIN</td>
<td>NSGA-II vs RS</td>
<td>&lt;0.001 [0.90]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.89]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs MOCell</td>
<td>0.011 [0.75]</td>
<td>0.195 [0.63]</td>
<td>0.148 [0.62]</td>
<td>&lt;0.001 [0.88]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs SPEA2</td>
<td>0.019 [0.75]</td>
<td>0.039 [0.63]</td>
<td>0.077 [0.65]</td>
<td>0.007 [0.87]</td>
</tr>
<tr>
<td>USERGRID</td>
<td>NSGA-II vs RS</td>
<td>&lt;0.001 [0.96]</td>
<td>&lt;0.001 [0.94]</td>
<td>&lt;0.001 [0.94]</td>
<td>&lt;0.001 [0.98]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs MOCell</td>
<td>&lt;0.001 [0.83]</td>
<td>&lt;0.001 [0.75]</td>
<td>&lt;0.001 [0.82]</td>
<td>&lt;0.001 [0.90]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs SPEA2</td>
<td>&lt;0.001 [0.82]</td>
<td>&lt;0.001 [0.75]</td>
<td>&lt;0.001 [0.83]</td>
<td>&lt;0.001 [0.90]</td>
</tr>
<tr>
<td>MESOS</td>
<td>NSGA-II vs RS</td>
<td>&lt;0.001 [0.96]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.95]</td>
<td>&lt;0.001 [0.96]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs MOCell</td>
<td>&lt;0.001 [0.86]</td>
<td>&lt;0.001 [0.70]</td>
<td>&lt;0.001 [0.83]</td>
<td>&lt;0.001 [0.88]</td>
</tr>
<tr>
<td></td>
<td>NSGA-II vs SPEA2</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.73]</td>
<td>&lt;0.001 [0.87]</td>
<td>&lt;0.001 [0.88]</td>
</tr>
</tbody>
</table>

RS: Random Search

### Table 3.5:  Comparison of MOSBIP vs. GA-BV and GA-GoA using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets).

<table>
<thead>
<tr>
<th>Project</th>
<th>MOSBIP vs</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLIDER</td>
<td>GA-BV</td>
<td>&lt;0.001 [0.84]</td>
<td>&lt;0.001 [0.76]</td>
<td>&lt;0.001 [0.80]</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>&lt;0.001 [0.84]</td>
<td>&lt;0.001 [0.62]</td>
<td>&lt;0.001 [0.77]</td>
</tr>
<tr>
<td>MAHOUT</td>
<td>GA-BV</td>
<td>0.687 [0.67]</td>
<td>0.03 [0.85]</td>
<td>0.031 [0.66]</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.02 [0.88]</td>
<td>0.031 [0.83]</td>
<td>0.031 [0.83]</td>
</tr>
<tr>
<td>AURORA</td>
<td>GA-BV</td>
<td>&lt;0.001 [0.84]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.81]</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>&lt;0.001 [0.90]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.85]</td>
</tr>
<tr>
<td>KYLIN</td>
<td>GA-BV</td>
<td>0.195 [0.63]</td>
<td>0.015 [0.75]</td>
<td>0.015 [0.75]</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>0.015 [0.75]</td>
<td>0.015 [0.87]</td>
<td>0.015 [0.75]</td>
</tr>
<tr>
<td>USERGRID</td>
<td>GA-BV</td>
<td>&lt;0.001 [0.83]</td>
<td>&lt;0.001 [0.77]</td>
<td>&lt;0.001 [0.85]</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>&lt;0.001 [0.83]</td>
<td>&lt;0.001 [0.83]</td>
<td>&lt;0.001 [0.88]</td>
</tr>
<tr>
<td>MESOS</td>
<td>GA-BV</td>
<td>&lt;0.001 [0.93]</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.88]</td>
</tr>
<tr>
<td></td>
<td>GA-GoA</td>
<td>&lt;0.001 [0.94]</td>
<td>&lt;0.001 [0.70]</td>
<td>&lt;0.001 [0.85]</td>
</tr>
</tbody>
</table>

MOSBIP: our approach using NSGA-II;
### Table 3.6: Comparison of MOSBIP against Random Guessing (RG) using Wilcoxon test and $\hat{A}_{XY}$ effect size (in brackets).

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLIDER</td>
<td>MOSBIP vs. RG</td>
<td>&lt;0.001 [0.93]</td>
<td>&lt;0.001 [0.92]</td>
<td>&lt;0.001 [0.93]</td>
</tr>
<tr>
<td>MAHOVB</td>
<td>MOSBIP vs. RG</td>
<td>&lt;0.001 [0.85]</td>
<td>&lt;0.001 [0.88]</td>
<td>&lt;0.001 [0.84]</td>
</tr>
<tr>
<td>AURORA</td>
<td>MOSBIP vs. RG</td>
<td>&lt;0.001 [0.93]</td>
<td>&lt;0.001 [0.96]</td>
<td>&lt;0.001 [0.96]</td>
</tr>
<tr>
<td>KYLIN</td>
<td>MOSBIP vs. RG</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.90]</td>
</tr>
<tr>
<td>USERGRID</td>
<td>MOSBIP vs. RG</td>
<td>&lt;0.001 [0.95]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.94]</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOSBIP vs. RG</td>
<td>&lt;0.001 [0.95]</td>
<td>&lt;0.001 [0.96]</td>
<td>&lt;0.001 [0.96]</td>
</tr>
</tbody>
</table>

The strategy of our MOSBIP approach provides a generic genetic variation in the natural population (i.e. maintains as good population diversity). Indeed, this search strategy provides a higher exploratory capability aiming to reach the optimal solution (i.e. the most suitable and fitted solution). Thus, the main reason of the enhancement lies in that MOSBIP combines the collective improving information of an offspring set and the distribution of current population without ignoring any objective then generates the evolving directions, rather than investigating a single objective. To investigate the reason behind the improvement by using our multi-objective search-based approach, we performed an extensive evaluation of 233 iterations from six large open source projects. The results we obtained demonstrate that our approach significantly outperforms random guessing in all projects. The evaluation also demonstrates the advantages of using a multi-objective approach over single-objective approaches. We also tested our approach with several multi-objective evolutionary algorithms (Random Search, MOCell and SPEA2) and found that NSGA-II was generally the best performers in this context.

#### 3.3.4 Threats to Validity

To mitigate conclusion validity threats, we carefully followed recent best practices to minimize conclusion instability \[\text{[208, 327, 328]}\]. We selected unbiased error measures and applied a number of statistical tests to verify our assumptions \[\text{[329]}\]. Our study
was performed on six datasets of different sizes. Furthermore, we applied the widely used metrics (precision, recall, and F-measure) to evaluate the effectiveness of our proposed approach. These metrics have been used in several software engineering studies \cite{308,330,331}. We also applied the hypervolume measure since it has been successfully used in related research work \cite{332}. Random initialization can also be subjected to another threat. Therefore, the results can be affected by either the favorable initial random selection or the bad randomly selected point if a single run of an experimental study carried out \cite{332}. To avoid this problem, we have conducted multiple runs (30 runs in this study) and chose the mean result.

To overcome the \textit{external validity} threat, we have considered 233 sprints and 55662 issues from six large open source projects. The size and the complexity of these issues are also significantly diverse. All sprints and associated issue reports are real data. These data are generated from open source settings in software development. However, we cannot claim that our datasets are representative of all kind of software projects, and that our results can generalize to all software projects especially those in commercial settings.

\section{Related Work}

There has been a range of work (i.e. \cite{130,333}) in dealing with the problem of selecting an optimal next release. Some of them employed multi-objective evolutionary techniques to select an optimal set of requirements for the next release. Our work is inspired by the notion and ideas of the next release problem. However, we specifically focus on agile settings in which our model explicitly captures important elements that are specific to agile development. For example, existing work in the next release problem literature did not take the goal of an iteration into consideration nor the team’s capability (velocity).
Several existing work \[334, 335\] have addressed the sprint planning problem in agile settings, but they did not use a multi-objective evolutionary approach. Our work is also related to bug prioritization (e.g. \[336, 337\]), but our approach captures not only the priority of issues but also other important elements such as goal alignment. Machine learning techniques have also been used to provide analytics support in agile development. For example, Choetkirtikul et al. \[308\] proposed an automated approach to support the project managers and other decision makers in predicting delivery capability for ongoing iterations, while several other works (e.g. \[307, 338\]) provide automated support for estimating story points.

3.5 Chapter Summary

We have proposed a multi-objective search-based approach (MOSBIP) to support the team in select issues for an upcoming iteration, as part of iteration planning in agile software development settings. Our approach leverages a meta-heuristic technique, namely genetic algorithms to search for subsets of issues in the product backlog. The search is guided simultaneously by two objectives: maximize the business value that an iteration delivers to the customers, and maximize the alignment of the selected issues with the iteration’s goal.

An extensive evaluation performed on 233 iterations from six large open source projects demonstrates that our approach significantly outperforms random guessing in all projects. The evaluation also demonstrates the advantages of using a multi-objective approach over single-objective approaches. We also tested our approach with several multi-objective evolutionary algorithms and found that NSGA-II was generally the best performers in this context. Future work would involve validating these results with additional projects, especially those in the commercial settings. We also plan to investigate the use of other objectives, especially those that capture the dependencies
between issues. We will also explore the use of other multi-objective evolutionary algorithms as part of our future work. In the next chapter, we propose an approach to predict issue effort and resolution time.
Chapter 4

Issue effort and time estimation

Predicting issue resolution time and effort is an important aspect of today’s software project management in terms of both development management and user management. In a software project, an issue represents description of a bug or a security vulnerability (e.g. bug report issues), or a description of a new functionality (e.g. feature request or user story issues) or enhancements of an existing functionality, or a project task. Most of today’s software projects use issue tracking systems (e.g. JIRA) which record those requests as issues. A software project therefore consists of a number of past issues (i.e., issues that have been closed), ongoing issues (i.e., issues that the team is working on), and new issues (i.e., issues that have just been created). Large long-term software projects may have tens of thousands of issues.

Knowing when an issue would be resolved is important for many stakeholders. For example, the resolution time from when an issue is reported until it is resolved is important from the user management’s point of view. The end-users are usually interested in knowing when the new functionality they requested is ready to be implemented. The bug reporters may be interested in learning when a particular bug is fixed. This resolution time can however be different from the actual effort for resolving

[1] https://www.atlassian.com/software/jira
an issue. For example, the user may have to wait for few weeks for a small feature they have requested to be implemented although it took the developers only a few hours to do this. Story points are the most common unit of measure used for estimating the effort involved in completing an issue. Those estimated efforts are used for by different stakeholders as input for prioritizing issues, planning and scheduling for future iterations and releases, costing and allocating resources, and tracking a team’s progress rate. Predicting either when a particular issue is going to be resolved or the effort of resolving it are both important and challenging. Existing practices in the industry often use the average resolution time or effort of past issues, combined with a certain margin of error, as an estimate for the resolution time/effort of the new issues. However, software issues may be significantly different from one another in their nature and the complexity of resolving them. Hence, the quality of the average resolution time/effort as an estimator is often poor. Other existing practices heavily rely on experts’ (e.g. project managers or experienced developers) subjective assessment to arrive at an estimate for the time and effort of resolving an issue. Relying on expert knowledge is however sometimes based on outdated experience and an underlying bias, thus may lead to inaccuracy in estimation.

Machine learners have also widely used for both estimating the time and effort required for resolving issues in software project. These work (e.g., [194,307,338–348]) mine the historical data generated when issues were reported and resolved. They identify features which characterize an issue and also influence on its resolution time. They then build machine learning models, train them using historical issues with known resolution time, and use them for future estimations.

In this chapter, we propose an alternative, multi-objective search-based evolutionary approach to predict the effort (in terms of both resolution time and story points) of each single issue in a software project. Specifically, we leverage a meta-heuristic
technique, namely genetic programming (GP)\cite{100}, to generate a large number of candidate estimation models, and search for the ones that are optimal with respect to a number of objectives. We explore two objectives guiding our search algorithms. The first objective is to minimize the Sum of Absolute Errors, which measures the accuracy of an estimation model in terms of the differences between values (i.e. issue resolution time and/or story points) estimated by the model and the values actually observed. The pressure of minimizing the estimation errors may, however, cause the solution model to adhere precisely to noisy data in the training set, which potentially make the model be excessively large and complex (hence, overfitting problems). While accuracy is critical for an estimation model, the Occam’s Razor principle of parsimony also plays an important role here: the model needs to be expressed in a simple way, easy for software practitioners to interpret\cite{349}. Hence, our second objective is to minimize the complexity of an estimation model, which can be measured in terms of the size of an expression tree representing the model. This second objective also leads to reduced computational costs since it encourages parsimonious (thus, computationally efficient) candidate solutions be generated. We name our approach Multi-Objective Issue Effort Estimation (MOIEE). This chapter makes a significant contribution in the following aspects:

- We address the broad context of software effort estimation from both the development and user management perspectives. Thus, our approach is able to predict not just only the resolution time of resolving an issue but also the effort of resolving it in story points. To the best of our knowledge, our work is the first one which studies both of those effort aspects at the issue level.

- We have employed two additional state-of-the-arts multi-objective evolutionary search algorithms: S-Metric Selection Evolutionary Multiobjective Optimiser Algorithm (SMS-EMOA)\cite{166} and the Indicator-Based Evolutionary Algorithm
This allows us to observe how a wide range of optimization algorithms perform in predicting the effort of resolving issues in software projects.

- Three standard performance measures, namely Median Absolute Error (MdAE), Mean of the Magnitude of Relative Error (MMRE), and Hypervolume, have been used in our evaluation.

- We developed eight additional projects for evaluation which has 12,937 issues across 13 large open source projects.

- The experimental evaluation have been significantly extended and redone entirely due to the above substantial extensions, and new results are reported in this chapter.

Our approach outperforms the three common baselines (random guessing, and mean and median), state-of-the-art techniques (linear regression, case-based reasoning, and random forests), and other two different multi-objective optimization algorithms (S-metric selection evolutionary multi-objective algorithm (SMS-EMOA) and the indicator-based evolutionary algorithm (IBEA)). The evaluation was performed against two different datasets. In terms of the resolution time estimation, we collected 8,260 issues from five different projects including four Apache Hadoop projects (Common, HDFS, MapReduce, Yarn) and one Apache Mesos project. In terms of the story points estimation, from eight different projects (APSTUD, MULE, DNN, TIMOB, MESOS, TISTUD, XD, and NEXUS) we collected 4,677 issues.

To evaluate the performance of estimation models, we use four standardized measures, Mean Absolute Error (MAE), Mean Magnitude of Relative Error (MMRE), Median Absolute Error (MdAE), and Standardized Accuracy (SA), and also use a non-parametric Wilcoxon test and Vargha and Delaney’s statistic to demonstrate both the statistical significance and the effect size of the results.
The remainder of this chapter is organized as follows. Section 4.1 illustrates an example to motivate our work. Section 4.3 describes our multi-objective approach to solve this problem using evolutionary algorithms. Section 4.4 reports on the experimental evaluation of our approach. Related work is discussed in Section 4.5 before we conclude and outline future work in Section 4.6.

### 4.1 Motivation example

An issue represents a request of fixing a bug, or a description of new functionality request, or a request for adapting an existing functionality to a new environment. Figure 4.1 shows the report of issue MESOS-4985 in the Apache Mesos project. This issue was a bug regarding destroying a container. This issue report was recorded in the widely-used JIRA project management system.

![Figure 4.1: A Motivation example of an issue with estimated resolution time and story points.](https://issues.apache.org/jira/browse/MESOS-4985)
Software effort estimation can be assessed from two different perspectives: developer management and user management. From the point of view of the developer management, the effort of resolving an issue would be a good measure since it reflects how easy it is to modify the software including how much work needed to be done, the complexity of the work, and any uncertainty involving in the work. Estimating the effort of completing each individual issue has thus become popular in the software maintenance practice (as opposed to the traditional effort estimation for the entire new project). Predicting this effort help the team prioritize user stories, plan and schedule for future iterations and releases, and even perform costing and allocating resources.

In many today’s agile projects, story points are commonly used as a unit of effort measure for each issue. For example, issue MESOS-4985 was estimated as 3 story points by the Apache Mesos team (see Figure 4.1). Story points are relative values: an issue that is assigned 3 story points should take triple as much effort as an issue assigned 1 story point. Story points are thus can be used to compare the effort of one issue to another in the same project. Existing practices in story point estimation (e.g. planning poker and analogy) require manual assessment from the team members. In a large software projects with a substantial number of issues, it is highly challenging for a team to be consistent in their story point estimates (to avoid reducing the predictability in planning and managing their project).

The end users perceive the effort estimation of a software from a different perspective. They are specifically concerned with the issue resolution time, i.e. the resolution time from when an issue was first reported until it was resolved. This time is, in many cases, different from the effort of resolving the issue. For example, although it might take only a few hours for resolving issue MESOS-4985, its resolution time was 5 days (reported on 18 March 2016 and resolved on 23 March 2016) since the issue might
be put on hold for a few days. Estimating when an issue would be resolved is highly
difficult, and thus a common approach in practice using the average resolution time
of past issues. Since the issues may vary substantially in nature and complexity, this
ad-hoc approach often gives inaccurate estimations, resulting in being unable to meet
the expectation from the end users.

While estimating the effort of resolving an issue is important for development
planning, being able to estimate when an issue is going to be resolved is critical in
managing user expectation and satisfaction. We formulate both of these estimations
as a multi-objective optimization problem and employ an evolutionary approach to
develop machinery which helps the team to predict the effort (in terms of both reso-
lution time and effort) for an issue in a software project. This machinery relies on a
set of information associated with an issue which we are going to describe next.

4.2 Issue features

4.2.1 Title and Description

The title and description of an issue explains its nature and thus can be a good
feature to be used by developers for effort estimation. Figure 4.1 shows an example
of issue MESOS-4985 in the Mesos project which is recorded in JIRA. An issue has
a title (e.g."Destroy a container while it’s provisioning can lead to leaked provisioned
directories") and description. Both issue title and description are written as natural
language texts. There are several widely-used robust approaches to translate them
into a set of features that are computationally convenient. A common approach is to
translate an issue’s title and description into the number of word counts. Readability
of the issue description as another feature. Readability is a quality indicator for issue
reports, and has been used in several studies [350–354]. We hypothesize that issues
that are more difficult to understand are going to be more difficult to deal with and thus potentially take longer time to resolve. We use the Gunning fog readability metric \[355\] to measure an issue description’s readability score. The lower Gunning fog score is, the easier to understand an issue description. The formula of the gunning fog is defined as follows \[354\]:

\[
GunningFog(d) = 0.4 \times \left( \text{AvgSL}(d) + \text{HWP}(d) \right)
\]

(4.1)

Where \(i=1\) \(d\) is issue description, \(\text{AvgSL}\) is the average sentence length in issue description and \(\text{HWP}\) is the Hard words’ percentage (i.e. the words which include more than two syllables) in issue description.

Another approach is combining and description of an issue report into one single text. Since the description of an issue may contains code snippets, natural language description and code snippet are separately processed using Term Frequency-Inverse Document Frequency (TF-IDF) techniques. A TF-IDF matrix is obtained form the natural language description and another TF-IDF matrix for the code snippet, both of which are then combined to form a set of features of an issue.

### 4.2.2 Issue Type

Each issue is assigned with a type (e.g. Bug, Task, Improvement, New feature, etc.) which indicates the nature of the task associated with resolving with the issue. For example, a “bug” issue reports a defect while a “new feature” describes a request for implementing new functionality. Figure 4.1 shows an example of an issue type of issue MESOS-4985 in Mesos project (recorded in JIRA) which is assigned as a bug. Since the issue type is categorical feature, we need to perform additional steps to ensure the results from a regression model be interpretable. Specifically, we use one hot encoding to transform the issue type in a number of features (corresponding to the number of
issue types), each of which represents one type and has a value of either 0 or 1. We used this feature for both resolution time, and the story points estimations.

### 4.2.3 Priority

The issue’s priority presents the order in which an issue should be attended with respect to other issues. In the projects we studied, there are 5 common types of priority: Blocker, Critical, Major, Minor, and Trivial. Issues with blocker priority should be more concerned than issues with major or minor priority since the former block other issues to be completed. Figure 4.1 illustrates an example of an issue’s priority which is assigned as a critical of issue MESOS-4985 in Mesos project. For issue priority, we convert it into an ordinal value from 1 to 5 where 1 represents the least priority and 5 represents the most priority. We used this feature for resolution time estimation.

### 4.2.4 Reporter

We use reporter’s reputation, a common feature which has been studied in previous work in mining bug reports [308, 344, 350]. The intuition here is that poor issue reports may take longer time to resolve and reporters who frequently write them would accumulate such a reputation. We use the widely-used Hooimeijer’s reporter reputation [350] as follows:

\[
reputation(D) = \frac{|opened(D) \cap fixed(D)|}{|opened(D)| + 1}
\] (4.2)

The reputation of a reporter $D$ is measured as the ratio of the number of issues that $D$ has opened and fixed to the number of issues that $D$ has opened plus one. We used this feature for resolution time estimation.
4.2.5 Components

As a categorical feature, components are a subset of the project (i.e., smaller pieces of the software or system) where issues can be grouped into smaller parts to form an issue report. The component can be considered as important information to be mainly used by the developers. Note that, after creating the component/s, an issue can be specified to which component it belongs to and hence that assists to define a scope of the issue (i.e. the level of issue/task). For example, determining the relevant parts in the software codebase to be modified. An example of this feature is shown in Figure 4.1. It can be seen that issue MESOS-4985 in project Mesos, this component is assigned with “containerization”. Similarly to issue type feature, we represent the components fields with a binary rating (i.e., matching components are rated with a value of 1 and the value of 0 for otherwise). We used this feature for story points estimation.

4.3 Approach

This section first presents our search-based approach including an overview of our approach, symbolic regression where estimation model is represented as an expression tree, fitness function, and the evolutionary search.

4.3.1 Overview

Our approach falls under the search-based software engineering umbrella. Figure 4.2 gives an overview of our approach. We build a training set by collecting completed issues from a given project and extracting their actual resolution time or story points. We design a set of features characterizing an issue (see Section 4.2) and extract the values of these features. We then iteratively generate candidate estimation models (by
using a set of mathematical operators to combine the issue features) and search for the “best” estimation model with respect to the training set. This search process employs evolutionary algorithms which work based on the principle that a population of candidate solutions (also referred to as individuals) to an optimization problem is evolved toward better solutions, following Darwin’s evolution theory. Each candidate solution has a number of properties (i.e., chromosomes or genotype) which can be mutated and altered to derive new candidate solutions. In our context of effort estimation in terms of resolution time and story points, a candidate solution is an estimation model.

The estimation model found at the end of the search process is used for predicting the effort of new issues in the same project (within-project estimation) or in a different project (cross-project estimation). We are going to describe the details of our approach.
4.3.2 Symbolic regression

Effort estimating (resolution time and story points) at an issue level can be considered as a regression problem: we need to model the relationship between the effort of an issue (in either story points or resolution time) and a set of features characterizing the issue. Here, we estimate the issue resolution time in terms of continuous number of days and the effort of resolving an issue in terms of story points. For each estimation, we rely on a number of basic information associated with an issue: type, components, priority, reporter, summary and description (see Section 4.2) to extract the features. Since the issue type and component are categorical features, we need to perform additional steps to ensure the results from a regression model be interpretable. Specifically, we use one hot encoding to transform the issue type and components in a number of features (corresponding to the number of issue types and components), each of which represents one type and has a value of either 0 or 1. For issue priority, we convert it into an ordinal value from 1 to 5 where 1 represents the least priority and 5 represents the most priority.

An estimation model can be viewed as a mathematical expression which combines those features of an issue and a set of mathematical operators to output a scalar value representing the time required for resolving an issue or the story points assigned for an issue. We use a training dataset of past issues (with known resolution time and story points) to search the space of those mathematical expressions by our proposed MOIEE simultaneously with respect to a number of objectives to find the estimation model that best fits the training dataset (i.e. optimal model). Based on the trained dataset, the obtained model is then used to predict the resolution time or story points required for new issues. This approach is commonly referred to as symbolic regression and has been previously used by [14] in estimating effort for the whole project. We adopted this approach from [14], but also made several key differences: (i) not imposing a
4.3. Approach

fixed structure nor depth on candidate models; (ii) using a different second objective function to explicitly control the model’s complexity; and (iii) using a wider range of mathematical operators.

Since each candidate estimation model is a mathematical expression; we represent it as an expression tree to facilitate the application of genetic operators (described in details later) to derive new candidates. Issue features are encoded as leaves of the tree and mathematical operators as its internal nodes. We thus use genetic programming [100], a meta-heuristic algorithm in the family of evolutionary algorithms, which specifically deals with tree representation. We employ a wide range of thirteen mathematical operators (+, −, *, /, exp, log, log10, sin, cos, tan, power, square, squareroot).

Figure 4.3 shows an example of an expression tree representing a candidate estimation model in which our estimation (i.e. issue resolution time or story points) is calculated as:

$$EstimationModel = (\cos(f_1) + (f_2 \times \exp(f_3))) - \sqrt{\log(f_4) - \sin(f_5)}$$

where $f_1, f_2, f_3, f_4$ and $f_5$ are some of the thirteen issue features.

**Figure 4.3:** An example of expression tree representing a candidate estimation model. Note that $f_i$ represents a feature of an issue.
4.3.3 Fitness functions

The search for the best estimation model is guided by a number of fitness functions, which are used to compare if a candidate model is “better” than another one. For a target problem to be solved, each individual solution includes all the decision variables (i.e. the information needed to select a more or less efficient solution) to this problem. For each solution, a fitness function (i.e. a measure of performance) needs to be defined. Such fitness function guides the search technique to find the preferable solution (i.e. compare if a solution is better than another one). After that, based on the fitness function of each individual, a mechanism of selection is applied to decide the fittest individuals to be selected. We employ two fitness functions: one reflecting the accuracy of an estimation model in predicting effort and the other representing the model’s complexity. The details of these fitness functions are described as below.

4.3.3.1 Sum of Absolute Errors

We use a training set of past issues (with known resolution time and story points) to evaluate the accuracy of a candidate estimation model. A number of measures have been used to evaluate the predictive performance of an estimation model and can be classified in two groups: relative measures such as Mean of Magnitude of Relative Error or Root Mean Square Error (RMSE), or absolute measures like the Sum of Absolute Error (SAE). Previous work (e.g. [14,358]) have suggested that the predictive performance of estimation models found by genetic algorithms is affected by the use of those different measures as a fitness function. Specifically, using relative measures as a fitness function has a negative impact on the model accuracy, while absolute measures appear to not have damaging effect [14]. Hence, similarly to [14,195], we chose to use
the Sum of Absolute Error (defined below) as our first fitness function.

\[
SAE = \sum_{i=1}^{N} |ActualEE_i - EstimatedEE_i| \tag{4.3}
\]

where \(N\) is the number of issues in the training set, \(ActualEE_i\) is the actual effort estimation for issue \(i\) in the training set, and \(EstimatedEE_i\) is the estimated effort estimation by a candidate model.

4.3.3.2 Tree size

Using an accuracy measure as the sole fitness function may result in excessively complex estimation models. The pressure of minimizing the estimation errors may lead to solution models that "overfit" the training data, which thus negatively affects the generalization performance of the models. It also takes more computational resources to store and evaluate complex models during the evolution process. In addition, software practitioners usually find complex estimation models difficult to understand and interpret [349].

The simplest method to control the complexity of estimation models is imposing a fixed limit on the depth or size (i.e., the number of nodes) of expression trees representing those models. This approach however suffers from a number of limitations. Determining a good value for the limit is very challenging. A small limit may prevent good solutions from being generated, while a large limit may still result in overcomplex solutions. In addition, the process of eliminating non-conformance individuals from the population may create bias and adverse affect [359]. To control the balance between accuracy and complexity, we employ a second fitness function which measures the complexity of a solution estimation model. Since we represent an estimation model as an expression tree, the size of the tree can be used as a complexity indicator. Hence, the second fitness function returns the size of a solution tree, i.e., the number
of nodes in the tree. For example, there are 14 nodes in the tree in Figure 4.3, hence its size is 14. The tree size reflects to some extends both the depth and width of a tree.

### 4.3.4 Evolutionary search

The search for an estimation model starts with an initial population in which each individual in the population is a candidate estimation model. To evaluate the impact of a classifier, we tested our approach with three widely-used multi-objective optimization algorithm: the Non-dominated Sorting Genetic Algorithm (NSGA-II) [127], S-Metric Selection Evolutionary Multiobjective Optimiser Algorithm (SMS-EMOA) [166], and the Indicator-Based Evolutionary Algorithm (IBEA) [152].

The NSGA-II algorithm consists of the following steps (see Figure 4.4):

- Set the suitable inputs (i.e. parameters) such as population, the maximum number of generation, and crossover and mutation probabilities.

- Create the initial population by randomly generating a number of expression trees (each represents an individual) using the thirteen mathematical operators and several issue features.

- Compute the fitness values of each individual with respect to each of two fitness functions (see Section 5.2.5).

- Select individuals form parents; the population is then undergone a selection process to generate a new generation of individuals through the crossover and mutation operators. These genetic operators act directly on the expression trees to form new valid trees. The mutation operator chooses a node in the expression tree and substitutes the sub-tree at that node by a randomly generated sub-tree. Figure 4.6 shows an example of how the tree in Figure 4.3 is mutated.
crossover operators involve two parent trees and generate two offspring trees by exchanging selected branches (see Figure 4.5). Generated trees which give negative or invalid (e.g., division by zero) estimated story points are assigned with a very large (infinite) sum of absolute errors. This would prevent those trees from being selected in the evolution process. This evolution process continues until a fixed number of generations has been reached.

- Find a Pareto Front; we seek for estimation models that meet both objectives: high accuracy and low complexity. These solutions would belong to a Pareto front of estimation error and tree size (see Figure 4.7). To find such a Pareto front, we employ the NSGA-II algorithm which works based on the principle of non-dominated sorting (Pareto dominance). In multi-objective optimization, an individual is said to dominate another individual if the former is better than the latter with respect to at least one objective, and not worse in the remaining objectives. For example, in Figure 4.7 estimation model $E_1$ does not dominate $E_2$ since the former is smaller than the latter in tree size but has greater sum of absolute errors. On the other hand, $E_1$ dominates $E_4$ since the $E_1$ is smaller than $E_4$ in tree size and also has smaller sum of absolute errors.

- Compute the crowding distance; at each generation, NSGA-II sorts the current population into a number of non-dominated fronts (e.g., fronts 1, 2 and 3 in Figure 4.7). Each non-dominated front contains individuals which do not dominate each other. Individuals in the first non-dominated front dominate those in the second front, which in turn dominate individuals in the third front, and so on. Individuals in the same non-dominated front are assigned the same rank, which is the index of its front.

- Form offspring population (i.e., form a new parent in the next population); the
next generation is selected from a combination of the parent and offspring operation. This is to minimize the possibility of losing a high-quality solution. In the final generation, NSGA-II returns a set of non-dominated solutions. Choosing which one of these solutions to use is usually a user-specific decision. In our case, we choose to use the solution which has the lowest sum of absolute errors with respect to the training set. The NSGA-II is discussed in more details in the background chapter, Section 2.1.3.1.

The SMS-EMOA [166] is a steady-state algorithm which is applied to constant population size (i.e. the worst individual of the parent population is always replaced by the offspring). This strategy uses the non-dominated sorting (also used in the NSGA-II) for the ranking process and the hypervolume to remove individuals with the worst-ranked front (i.e. to select the non dominated individuals only). This algorithm generates only one new individual in each iteration. It thus updates population members (individuals) within a steady-state strategy. The SMS-EMOA starts with (step 1) generating a new individual from an initial population by the randomised variation operators (i.e. crossover and mutation). Afterwards (step 2), it combines the generated new individual with the current population to gain the next population. Then (step 3), the non-dominated sorting approach is applied to divide the next population and hence to obtain different non-domination levels (each level is called a front). After that (step 4), the selection process is applied where the hypervolume is used to remove individuals with the worst-ranked front. This procedure is repeated until the recommended population size is reached. For the detailed information on SMS-EMOA, we refer to the reader to [166].

The IBEA [152] strategy starts by (step 1) randomly generating a population P. Then (step 2), the fitness function is calculated for each solution (individual) from P. After that (step 3), an environmental selection process starts by which we select
individuals with the lowest fitness values from $P$ to be removed and hence the fitness values of the remaining population must be updated. *Step 4*, to be continued until the number of solutions in $P$ does not exceed population size. Then (*step 5*), we apply tournament selection operator to the population to start the mating selection to select two parents from $P$. Afterwards (*step 6*), we apply the crossover operator (on two parents to generate two offspring), and mutation operator (on one of the offspring). Hence, the obtained offspring are added to the population. This evolution process continues until a fixed number of generations has been reached. For more details on IBEA, we refer to the reader to [152]. To rank solutions, both SMS-EMOA and IBEA use the hypervolume performance while the NSGA-II algorithm uses the crowding distance technique.

![Non-dominated sorting genetic algorithm (NSGA-II) Flow chart](image)

**Figure 4.4:** Non-dominated sorting genetic algorithm (NSGA-II) Flow chart

Following the common practice [326], we used the following parameters. The size of the initial population is set to $100 \times V$ where $V$ is the number of features. The number of generations was set to $10,000 \times V$. Crossover probability was set to 0.9,
4.3. Approach

**Figure 4.5:** An example of the mutation operator

**Figure 4.6:** An example of the crossover operator
4.4 Evaluation

The evaluation aims to answer the following research questions:

RQ1. Sanity Check: Is the multi-objective search-based approach suitable to predict effort in terms of issue resolution time and effort in story points?

To answer this question, we compared our multi-objective search-based approach against three common baselines: Random Guessing, and Mean and Median. Random guessing is a naive technique for estimation \[14,208\]. It performs random sampling.

mutation probability was 0.1, and reproduction probability was 0.2. We used tournament selection method. These are common values used in previous studies \[14,326\]. We used the implementation of all three algorithms in the MOEA Framework\[3\].

\[14\] http://moeaframework.org/index.html

\[3\] http://moeaframework.org/index.html

Figure 4.7: An example of non-dominated fronts
over a set of issues with a known resolving time (or story points), choosing randomly one issue from the sample, and uses the resolving time (or story points) of that issue as the estimate of the target issue. Random guessing does not use any information associated with the target issue. Thus, any useful estimation model should outperform random guessing. Mean and Median estimations are also commonly used as baseline benchmarks for effort estimation. They use the mean or median resolution time (or story points) of the past issues to estimate the resolution time (or story points) of the target issue.

**RQ2. Different State-of-the-Art algorithms:** Does the search-based approach provide more accurate estimates than existing techniques used in predicting effort in terms of issue resolution time and story points?

To answer this question, we compare our search-based approach against the three existing techniques that have been widely used: Linear Regression (LR), Case-Based Reasoning (CBR) and Random Forests. For case-based reasoning, we used k-nearest-neighbour (kNN) as done in the seminal work [194]. Random Forests (RF) is chosen since it is currently the most effective method for effort estimation [301]. RF is an ensemble method which combines the estimates from multiple estimators. RF achieves a significant improvement over the decision tree approach by generating many classification and regression trees. Each tree is built on a random resampling of the data, with a random subset of variables at each node split. Then through averaging, tree predictions can be aggregated. Note that all these three prediction models use the same set of features as in our approach.

We used the implementation of linear regression and kNN provided with Weka. Since it is tedious to find the optimal hyperparameters for these classification or regression algorithms, we automated this process using Weka’s MultiSearch, a meta-classifier
for tuning hyperparameters of a given base classifier or regressor. Specifically, for linear regression, we focused on tuning two hyperparameters: ridge (ranging from 1e-7 to 10) and selection method (with three methods: none, greedy and M5). For kNN, we used the brute force search algorithm (i.e. LinearNNSearch), Euclidean distance for the distance function, and tuned \( k \) number ranging from 1 to 64. We experimented with two implementations of Random Forests: one provided in Weka and the other written in Matlab\(^4\), and found that the Matlab implementation produced better results with our data. Thus we chose this implementation to compare against our approach. We tuned Random Forests from 1 to 500 trees. All tuning was done using training data.

In addition, we also compare our approach (MOIEE) against two state-of-the-art techniques recently proposed by Porru et. al.\(^{338}\) and Choetkiertikul et. al.\(^{307}\) for story point estimation to examine the applicability of our obtained model using different dataset. Porru et. al. employed traditional machine learner while Choetkiertikul et. al. used a deep learning approach. Therefore, while our approach an explicit model which is able to explain its estimation, both Porru et. al.’s and Choetkiertikul et. al.’s models are “black box” in nature which offers limited explainability.

**RQ3. Different multi-objective optimization algorithms:** Which multi-objective optimization algorithms perform best with our approach?

Our approach is generic in which different multi-objective optimization algorithms can be used. Although NSGAII is the main algorithm (described in details in Section 4.3 that we employed, there are also other suitable algorithms for our approach. In our extended experimental work, We have tested our approach with two recently developed multi-objective evolutionary algorithms: S-metric selection evolutionary multi-objective algorithm (SMS-EMOA)\(^{166}\) and the indicator-based evolutionary algorithm (IBEA)\(^{152}\).

\(^4\)https://github.com/ami-GS/randomforest-matlab
4.4. Evaluation

RQ4. Benefits from Multi-objective Approach: *Does the multi-objective approach produce more accurate estimates than the single-objective approach in predicting effort regarding resolution time and story points, and at the same time produce solutions of lower complexity?*

This question aims to investigate whether using the tree size as the second objective offers significant benefits in terms of both accuracy and complexity. To do so, we also implemented the traditional single-objective genetic programming algorithm using the sum of absolute error as the objective function. We name this alternative approach as GP-SAE. We experimented with two variants of this algorithm, one with unlimited depth tree and the other with a limited depth tree of 10. The former represents a method with no control on the complexity of the solutions, while the latter represents a technique with a constant limit on the tree size.

RQ5. Cross-Project Estimation: *Is the proposed approach suitable for cross-project estimation?*

Estimating whether resolution time or story points in new projects is often difficult due to the lack of training data. One common technique can be applied to deal with this problem which is training a model using data from a (source) project and applying it to the new (target) project. We employed this technique and performed 20 (resolution time), and 56 (story points) cross-project estimation experiments.

We now describe how data was collected and preprocessed for our evaluation. We then discuss the experimental settings and performance measures before discussing the evaluation results.

4.4.1 Datasets

In this section, we describe how data were collected for our empirical study and the experiments.
4.4. Evaluation

4.4.1.1 Data collecting and pre-processing

We built a dataset for resolution time from five different projects including four Apache Hadoop\textsuperscript{5} projects (Common, HDFS, MapReduce, Yarn) and one Apache Mesos\textsuperscript{6} project, from the well-known Apache to build a dataset of issues with the known creation and resolved times. Those issues were recorded in the widely-used JIRA tracking system. We used the Representational State Transfer (REST) API provided by JIRA for querying the issues. After that, we collected issue reports in JavaScript Object Notation (JSON) format. The collected issues have the created date and resolved date up to September 9, 2016, and for Mesos is up to March 24, 2017. From these projects, we extracted a total of 16,858 issues. We excluded issues with a status other than “Resolved” to avoid collecting uncompleted issues and a resolution other than “Fixed” (e.g. “Duplicate” or “Not Fix”, or “Invalid”) in order to collect only “real” issues. In the end, we included 8,260 issues into our dataset. We have calculated the duration time for each issue by subtracting its resolved time from its created time. The resolution time is measured in days. Table \ref{tab:stats} summarizes the descriptive statistics of our dataset in terms of mean, median, mode and standard deviation of resolution time. The median resolution times range from 7 to 18 days in the five projects, while the mean resolution times were from 28 to 52 days. Although the resolution times varied quite substantially (standard deviation in all projects were above 55), most of the issues were resolved within 2 days.

For story points, we used a publicly available \cite{338} dataset which contains eight projects, namely Aptana Studio (APSTUD)\textsuperscript{7}, Mule (MULE)\textsuperscript{8}, Dnn Platform (DNN)\textsuperscript{9}, Titanium SDK/CLI (TIMOB)\textsuperscript{10}, Apache Mesos (MESOS), Appcelerator Studio (TIS-

\footnotesize\textsuperscript{5}http://hadoop.apache.org/
\textsuperscript{6}http://mesos.apache.org/
\textsuperscript{7}http://www.aptana.com/
\textsuperscript{8}https://www.mulesoft.com/
\textsuperscript{9}http://www.dnnsoftware.com/products
\textsuperscript{10}https://jira.appcelerator.org/browse/TIMOB
4.4. Evaluation

Table 4.1: Descriptive statistics of the resolution time for the projects in our datasets

<table>
<thead>
<tr>
<th>Projects</th>
<th># selected issues</th>
<th>Mean</th>
<th>Median</th>
<th>Mode</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON</td>
<td>1,402</td>
<td>37.19</td>
<td>10</td>
<td>2</td>
<td>62.24</td>
</tr>
<tr>
<td>HDFS</td>
<td>2,334</td>
<td>28.66</td>
<td>7</td>
<td>2</td>
<td>55.49</td>
</tr>
<tr>
<td>MAPREDUCE</td>
<td>635</td>
<td>45.68</td>
<td>14</td>
<td>2</td>
<td>72.62</td>
</tr>
<tr>
<td>YARN</td>
<td>930</td>
<td>46.82</td>
<td>17</td>
<td>2</td>
<td>69.32</td>
</tr>
<tr>
<td>MESOS</td>
<td>2959</td>
<td>52.97</td>
<td>18</td>
<td>1</td>
<td>77.13</td>
</tr>
<tr>
<td>Total</td>
<td>8,260</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.2: Descriptive statistics of the story points for the projects in our datasets

<table>
<thead>
<tr>
<th>Project</th>
<th># Issues</th>
<th>Min/Max</th>
<th>Mean</th>
<th>Median</th>
<th>Mode</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>228</td>
<td>1/100</td>
<td>7.99</td>
<td>8</td>
<td>8</td>
<td>7.57</td>
</tr>
<tr>
<td>DNN</td>
<td>864</td>
<td>1/13</td>
<td>1.91</td>
<td>2</td>
<td>1</td>
<td>1.25</td>
</tr>
<tr>
<td>MESOS</td>
<td>353</td>
<td>1/13</td>
<td>2.79</td>
<td>2</td>
<td>1</td>
<td>2.14</td>
</tr>
<tr>
<td>MULE</td>
<td>631</td>
<td>1/13</td>
<td>4.60</td>
<td>5</td>
<td>5</td>
<td>3.18</td>
</tr>
<tr>
<td>TIMOB</td>
<td>588</td>
<td>0.5/13</td>
<td>5.07</td>
<td>5</td>
<td>5</td>
<td>3.26</td>
</tr>
<tr>
<td>TISTUD</td>
<td>1171</td>
<td>1/20</td>
<td>5.42</td>
<td>5</td>
<td>5</td>
<td>2.56</td>
</tr>
<tr>
<td>XD</td>
<td>429</td>
<td>1/8</td>
<td>2.87</td>
<td>2</td>
<td>1</td>
<td>2.04</td>
</tr>
<tr>
<td>NEXUS</td>
<td>413</td>
<td>0.5/8</td>
<td>1.14</td>
<td>1</td>
<td>1</td>
<td>0.86</td>
</tr>
<tr>
<td>Total</td>
<td>4677</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

TUD)\(^\text{11}\), Spring XD (XD)\(^\text{12}\), and Sonatype’s Nexus (NEXUS)\(^\text{13}\) from the well-known Apache to build a dataset of issues with known story points. Initially, from these eight open source projects, we collected 4,677 issues. Those issues were recorded in the widely-used JIRA tracking system. The original Porru’s dataset counts 16,523 issues. Porru has filtered the dataset to be counting down to 5607 issues. To extract our dataset, we strictly followed Parue’s steps, but we have filtered out issues assigned with zero and greater than 100 story points. Hence, our total dataset counts 4677 issues. We excluded issues with a status other than “Resolved” to avoid collecting uncompleted issues and a resolution other than “Fixed” (e.g., Duplicate” or Not Fix”, or Invalid”) in order to collect only real” issues. To achieve a more stable and consistent estimations, for each issue report, we select the story points which have

\(^{11}\text{http://www.appcelerator.com/}^{12}\text{http://projects.spring.io/spring-xd/}^{13}\text{http://www.sonatype.org/nexus/}
been assigned only once (i.e., never been changed). Besides, issue type, description, summary, and components have never been updated after story points have been assigned. We assigned our story points values from 0.5 to 100 (see Table 4.2). These story points values based on Planning Poker cards set, i.e. 0.5, 1, 2, 3, 5, 8, 13 and so on [300, 360, 361].

In the end, we included 4,677 issues into our dataset ranging from 228 (APSTUD) to 1171 (TISTUD). Table 4.2 summarizes the descriptive statistics of our dataset in terms of number of issues, minimum, maximum, mean, median, mode, and standard deviation of story points. For example, the median story points range from 1 to 8 points in the eight projects, while the mean story points were from 1.15 to 7.6 points.

4.4.2 Experimental Settings and Measures

4.4.2.1 Performance Measures

For each project, we applied a cross validation process which is a widely employed technique to validate an estimation or prediction model. Specifically, we divided our dataset into 10 folds and applied cross-validation (i.e. used nine folds for training and the remaining one fold for testing) to reduce the estimation instability and bias. A single run of an experimental study may deliver results that can be affected either by the favorable initial random selection or the bad randomly selected point [332]. Therefore, to avoid this problem, we ran each algorithm 30 times and took the median result. Similarly to previous work in effort estimation (e.g. [14, 195, 208, 307, 340, 362, 367]), we employed widely-used standardised measures, Mean Absolute Error (MAE) [364], Median Absolute Error (MdAE) [364], and the Standardized Accuracy (SA) [208], to avoid the bias towards over or under estimation. They are defined as below.

\[
MAE = \frac{1}{N} \sum_{i=1}^{N} |ActualEE_i - EstimatedEE_i|
\]

(4.4)
4.4. Evaluation

where $N$ is the total number of issues used in the test set. Estimation models with lower MAE are better in terms of accuracy.

Standardized Accuracy (SA) measures how good an estimation model is with respect to random guessing:

$$SA = (1 - \frac{MAE}{MAE_{guess}}) \times 100$$

(4.5)

where $MAE_{guess}$ is the $MAE$ averaging a large number of random guesses. Estimation models with larger SA are more useful.

The median absolute error (MdAE) measures the median of the absolute errors between the actual value and the model estimation. Estimation models with lower MdAE are better in terms of accuracy:

$$MdAE = \text{Median}\{|Actual_i - Estimated_i|\}$$

(4.6)

Where $1 \leq i \leq N$.

We employ error calculation method namely Mean of the Magnitude of Relative Error (MMRE) for each project which has also been widely used in effort estimation [338, 368–373]. It is calculated as [364]:

$$MMRE = \frac{1}{N} \sum_{i=1}^{N} |ActualEE_i - EstimatedEE_i|/ActualEE_i$$

(4.7)

where $N$ is the number of issues in the test set, $ActualEE_i$ is the actual effort estimation for issue $i$, and $EstimatedSP_i$ is the estimated effort estimation by a candidate model.

We also use hypervolume [177, 209] as a quality indicator for the volume of the
space covered by the non-dominated solutions. This measure has been used in previous work (e.g. [14, 324, 325]) as a performance indicator for multi-objective optimization. Hypervolume is the only metric that has the capability to consider all three aspects (diversity, cardinality, and accuracy). It reflects the convergence and diversity of the solutions on a Pareto front (e.g., the higher hypervolume, the better performance). Hypervolume is described [374, 375] as Pareto compliant (i.e., dealing with approximation sets). More explicitly, hypervolume can assure that approximation set with a maximax quality value, includes all Pareto optimal solutions.

4.4.2.2 Statistical Test Approaches

In order to compare the performance of two models, we employ the Wilcoxon Signed Rank Test [309] to assess the statistical significance of the absolute errors produced from those estimation models. This non-parametric effect size measure is suitable for testing randomized algorithms in software engineering, especially in the context of effort estimation [14, 376]. The Wilcoxon Signed Rank Test does not assume a normal distribution in the data which is a safe test. The null hypothesis here is: “the performance provided by our approach is not different to those provided by alternative approaches”, which we work to reject this null hypothesis. We set the confidence limit at 0.05 (i.e. \( p < 0.05 \)). We then assessed whether the effect size is interesting by employing the correlated samples case of the Vargha and Delaney’s \( \hat{A}_{XY} \) non-parametric effect size measure [310]. The \( \hat{A}_{XY} \) measures the probability that the estimation model \( E_i \) achieves better accuracy (i.e., smaller absolute errors) than estimation model \( E_j \). We employ the statistical testing and effect size testing which can be defined as the following formula [310]:

\[
\hat{A}_{XY}(AE) = \frac{\#(X_{AE} < Y_{AE}) + (0.5 \times \#(X_{AE} = Y_{AE}))}{m} \quad (4.8)
\]
Where \( \#(X_{AE} < Y_{AE}) \) is the number of issues that the Absolute Error (i.e. AE) from model X less than from model Y, \( \#(X_{AE} = Y_{AE}) \) is the number of issues that the Absolute Error from model X equal to the model Y, and \( m \) is the number of issues.

### 4.4.3 Results

In this section, we present the results of our experimental evaluation\(^\text{14}\) to answer each research question we have previously outlined.

**RQ1: Sanity check**

*In terms of issue resolution time,* as can be seen from Table 4.3, our approach (MOIEE) produced better estimations in terms of MAE, MMRE, MdAE, and SA than the Mean, Median and Random guessing did. MOIEE consistently outperformed all the three baselines in all five projects. Averaging across all projects, MOIEE achieved an accuracy of 22.02 MAE, 1.24 MMRE, 6.09 MdAE and 62.21 SA, while the best of the baselines (Median) achieved 38.65 MAE, 2.53 MMRE, 20.23 MdAE, and 33.49 SA.

*In terms of story points estimation,* Table 4.4 shows the results obtained from MOIEE, and the two baseline methods (Mean and Median). The analysis of MAE, MMRE, MdAE, and SA indicates that the estimations achieved with our approach (MOIEE), are better than those achieved by using Mean, Median, and Random guessing estimates. MOIEE consistently outperforms all these three baselines in all eight projects.

Our approach improved between 42.28% (in project APSTUD) to 66.07% (in project NEXUS) in terms of MAE, 61.01% (in TISTUD) to 86.51% (in NEXUS) in terms of MMRE, 48.38% (in TISTUD) to 95.31% (in NEXUS) in terms of MdAE and 46.01% (in DNN) to 114.04% (in MULE) in terms of SA over the Mean method. Our

\(^{14}\)All the experiments were run on a Microsoft Windows 10 Home PC with an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz and 16.00 GB RAM.
MOIEE approach shows improvements over the Median method between 41.44% (in APSTUD) to 64.33% (in MESOS) in MAE, 42.50% (in TISTUD) to 75% (in NEXUS) in MMRE, 46.50% (in TIMOB) to 94.00% (in NEXUS) in terms of MdAE, and 42.64% (in DNN) to 101.99% (in MULE) in SA.

Table 4.6 (resolution time) and Table 4.5 (story points) show the results of the Wilcoxon test and the effect size to measure the statistical significance and effect size (in brackets) of the improved accuracy achieved by MOIEE over the three baselines. In all cases, our approach MOIEE significantly outperforms the baselines with $p < 0.001$ and effect sizes greater than 0.63 which can be considered as large effect size ($\hat{A}_{XY} > 0.6$), in all cases and for all three measures.

**Answer to RQ1:** Our proposed approach, MOIEE, outperforms the naive benchmarks in all open source projects in terms of issue resolution time and story points estimation, thus passing the sanity check required by RQ1.
### Evaluation

Table 4.4: Evaluation results of our approach MOIEE in terms of story point estimation, the baselines (Mean and Median) (the best results are highlighted in bold). MAE, MMRE and MdAE - the lower the better, SA - the higher the better.

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>MAE</th>
<th>MMRE</th>
<th>MdAE</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>MOIEE</td>
<td>2.02</td>
<td>0.27</td>
<td>1.57</td>
<td>57.65</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>3.50</td>
<td>1.17</td>
<td>3.10</td>
<td>30.29</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>3.45</td>
<td>0.77</td>
<td>3.00</td>
<td>31.43</td>
</tr>
<tr>
<td>DNN</td>
<td>MOIEE</td>
<td>0.41</td>
<td>0.14</td>
<td>0.31</td>
<td>78.00</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.84</td>
<td>0.67</td>
<td>0.91</td>
<td>53.42</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>0.83</td>
<td>0.54</td>
<td>1.00</td>
<td>54.68</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOIEE</td>
<td>0.51</td>
<td>0.32</td>
<td>0.34</td>
<td>85.44</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.50</td>
<td>1.08</td>
<td>0.89</td>
<td>42.55</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.43</td>
<td>0.59</td>
<td>1.00</td>
<td>43.71</td>
</tr>
<tr>
<td>MULE</td>
<td>MOIEE</td>
<td>1.41</td>
<td>0.41</td>
<td>0.71</td>
<td>60.66</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>2.57</td>
<td>1.28</td>
<td>2.74</td>
<td>28.34</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>2.53</td>
<td>1.34</td>
<td>3.00</td>
<td>30.03</td>
</tr>
<tr>
<td>TIMOB</td>
<td>MOIEE</td>
<td>1.23</td>
<td>0.40</td>
<td>1.07</td>
<td>74.91</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>2.55</td>
<td>1.40</td>
<td>2.15</td>
<td>49.45</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>2.50</td>
<td>1.03</td>
<td>2.00</td>
<td>50.41</td>
</tr>
<tr>
<td>TISTUD</td>
<td>MOIEE</td>
<td>0.82</td>
<td>0.23</td>
<td>0.44</td>
<td>85.01</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>2.11</td>
<td>0.59</td>
<td>2.46</td>
<td>53.75</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.88</td>
<td>0.40</td>
<td>2.00</td>
<td>57.61</td>
</tr>
<tr>
<td>XD</td>
<td>MOIEE</td>
<td>0.76</td>
<td>0.27</td>
<td>0.50</td>
<td>72.34</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.62</td>
<td>1.02</td>
<td>1.80</td>
<td>41.59</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.58</td>
<td>0.56</td>
<td>1.00</td>
<td>42.78</td>
</tr>
<tr>
<td>NEXUS</td>
<td>MOIEE</td>
<td>0.19</td>
<td>0.12</td>
<td>0.03</td>
<td>81.96</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.56</td>
<td>0.89</td>
<td>0.64</td>
<td>49.44</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>0.48</td>
<td>0.48</td>
<td>0.50</td>
<td>56.95</td>
</tr>
</tbody>
</table>

**RQ2: Different State-of-the-Art algorithms**

In terms of resolution time estimation, the MAE, MMRE, MdAE, and SA results (see Table 4.7) shows that Random Forests is the best performer amongst the three existing techniques. However, when comparing against our approach, Random Forests consistently produced higher (MAE, MMRE, MdAE) and lower SA than MOIEE in
### 4.4. Evaluation

**Table 4.5:** Comparison between our approach MOIEE and the three baseline techniques in terms of story points using Wilcoxon test and $\hat{A}_{xy}$ effect size (in brackets)

<table>
<thead>
<tr>
<th>MOIEE vs.</th>
<th>Mean</th>
<th>Median</th>
<th>Random Guessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>$&lt;0.001$ [0.70]</td>
<td>$&lt;0.001$ [0.64]</td>
<td>$&lt;0.001$ [0.87]</td>
</tr>
<tr>
<td>DNN</td>
<td>$&lt;0.001$ [0.76]</td>
<td>$&lt;0.001$ [0.67]</td>
<td>$&lt;0.001$ [0.92]</td>
</tr>
<tr>
<td>MESOS</td>
<td>$&lt;0.001$ [0.80]</td>
<td>$&lt;0.001$ [0.63]</td>
<td>$&lt;0.001$ [0.85]</td>
</tr>
<tr>
<td>MULE</td>
<td>$&lt;0.001$ [0.74]</td>
<td>$&lt;0.001$ [0.71]</td>
<td>$&lt;0.001$ [0.89]</td>
</tr>
<tr>
<td>TIMOB</td>
<td>$&lt;0.001$ [0.73]</td>
<td>$&lt;0.001$ [0.68]</td>
<td>$&lt;0.001$ [0.80]</td>
</tr>
<tr>
<td>TISTUD</td>
<td>$&lt;0.001$ [0.84]</td>
<td>$&lt;0.001$ [0.73]</td>
<td>$&lt;0.001$ [0.91]</td>
</tr>
<tr>
<td>XD</td>
<td>$&lt;0.001$ [0.71]</td>
<td>$&lt;0.001$ [0.66]</td>
<td>$&lt;0.001$ [0.87]</td>
</tr>
<tr>
<td>NEXUS</td>
<td>$&lt;0.001$ [0.86]</td>
<td>$&lt;0.001$ [0.65]</td>
<td>$&lt;0.001$ [0.93]</td>
</tr>
</tbody>
</table>

**Table 4.6:** Comparison between our approach MOIEE and the three baseline techniques in terms of elapsed time using Wilcoxon test and $\hat{A}_{xy}$ effect size (in brackets)

<table>
<thead>
<tr>
<th>MOIEE vs.</th>
<th>Mean</th>
<th>Median</th>
<th>Random Guessing</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON</td>
<td>$&lt;0.001$ [0.81]</td>
<td>$&lt;0.001$ [0.79]</td>
<td>$&lt;0.001$ [0.89]</td>
</tr>
<tr>
<td>HDFS</td>
<td>$&lt;0.001$ [0.86]</td>
<td>$&lt;0.001$ [0.83]</td>
<td>$&lt;0.001$ [0.92]</td>
</tr>
<tr>
<td>MAPREDUCE</td>
<td>$&lt;0.001$ [0.85]</td>
<td>$&lt;0.001$ [0.80]</td>
<td>$&lt;0.001$ [0.92]</td>
</tr>
<tr>
<td>YARN</td>
<td>$&lt;0.001$ [0.89]</td>
<td>$&lt;0.001$ [0.86]</td>
<td>$&lt;0.001$ [0.95]</td>
</tr>
<tr>
<td>MESOS</td>
<td>$&lt;0.001$ [0.79]</td>
<td>$&lt;0.001$ [0.75]</td>
<td>$&lt;0.001$ [0.93]</td>
</tr>
</tbody>
</table>

all five projects. The improvement brought by our approach over Random Forests was from 29.37% (in MESOS) to 46.29% (in Yarn) in MAE, 42.94% (in HDFS) to 52.40% (in COMMON) in MMRE, 68.07% (in MESOS) to 79% (in COMMON) in MdAE, 46.76% (in MESOS) to 78.40% (in Yarn) in SA. Overall, averaging across all five projects, MOIEE improved, in terms of MAE, 50.46% over LR, 54.87% over CBR, and 40.01% over RF.

*In terms of story points estimation,* Table 4.8 shows that our approach achieved arrange of improvement from 25.49% (in XD) to 46.87% (in MESOS) in MAE, 23.80% (in MESOS) to 64% (in APSTUD) in MMRE, 15.74% (in TIMOB) to 86.36% (in
NEXUS) in MdAE, and 10.80% (in TISTUD) to 75.67% (in MULE) SA over Random Forests. Using MOIEE improved over CBR between 28.11% (in APSTUD) to 63.30% (in MESOS) in MAE, 31.03% (in TIMOB) to 64.47% (in APSTUD) in MMRE, 25.23% (in APSTUD) to 86.95% (in NEXUS) in MdAE, and 11.92% (in DNN) to 85.27% (in MULE) SA. The improvements of our approach over the LR method are between 30.58% (in APSTUD) to 63.04% (in MESOS) in terms of MAE, 47.27% (in TISTUD) to 37.07% (in DNN) in terms of MMRE, 34.95% (in APSTUD) to 91.66% (in NEXUS) in terms of MdAE, and 25.32% (in TISTUD) to 91.35% (in MULE) in terms of SA.

In terms of story points estimation, our approach also outperforms both Choetkier-tikul et. al.’s [307] and Porru et. al.’s [338] models in all cases. In terms of MAE (see Table 4.9), MOIEE improved over Choetkiertikul et. al.’s model between 09.52% (in NEXUS) and 39.22% (in MULE). It also demonstrates a good improvement over Porru et. al.’s model between 30.11% (in TIMOB) and 64.49% (in APSTUD).

The Wilcoxon test (see Table 4.10 (resolution time) and Table 4.11 (story points)) also confirms this: the improvement of MOIEE over LR, CBR, and RF is significant ($p < 0.001$) in all cases (resolution time) and 20/24 cases (story points) with the effect size greater than 0.54 in all projects. These results suggest that our multi-objective search-based approach offers an alternative and effective technique to predict the effort (in terms of both resolution time and story points) of each single issue. The improvement may be due to its capability of capturing the nonlinear relationship between issue features and both resolution time and story points. Also, our approach does not carry any human biases nor affected by unknown domain-specific knowledge by not imposing any prior model structure and size.

**Answer to RQ2:** Our proposed approach using NSGA-II consistently out-performs state-of-the-art techniques in issue resolution time and story point estimation.
4.4. Evaluation

Table 4.7: Evaluation results of our approach MOIEE in terms of elapsed time, the state-of-the-art techniques: Linear Regression (LR), Case-Based Reasoning (CBR), and Random Forests (RF). The results for the single objective search with unlimited depth (GP-SAE) are also included – discussed later in RQ3 (the best results are highlighted in bold).

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>MAE</th>
<th>MMRE</th>
<th>MdAE</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON</td>
<td>MOIEE</td>
<td>20.66</td>
<td>0.99</td>
<td>3.99</td>
<td>60.37</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>27.55</td>
<td>1.61</td>
<td>5.42</td>
<td>47.15</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>40.68</td>
<td>3.18</td>
<td>29.32</td>
<td>21.99</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>44.63</td>
<td>3.39</td>
<td>23.00</td>
<td>14.42</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>33.89</td>
<td>2.08</td>
<td>19.00</td>
<td>35.02</td>
</tr>
<tr>
<td>HDFS</td>
<td>MOIEE</td>
<td>15.90</td>
<td>0.89</td>
<td>2.96</td>
<td>62.48</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>21.19</td>
<td>1.41</td>
<td>4.49</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>31.25</td>
<td>2.58</td>
<td>19.12</td>
<td>26.29</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>33.46</td>
<td>2.68</td>
<td>20.58</td>
<td>21.07</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>26.1</td>
<td>1.56</td>
<td>12.19</td>
<td>38.44</td>
</tr>
<tr>
<td>MAPREDUCE</td>
<td>MOIEE</td>
<td>21.16</td>
<td>1.32</td>
<td>6.65</td>
<td>66.72</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>33.57</td>
<td>1.92</td>
<td>9.72</td>
<td>47.21</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>50.46</td>
<td>3.96</td>
<td>37.94</td>
<td>20.67</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>58.06</td>
<td>4.25</td>
<td>30.73</td>
<td>8.72</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>39.38</td>
<td>2.54</td>
<td>27.8</td>
<td>38.1</td>
</tr>
<tr>
<td>YARN</td>
<td>MOIEE</td>
<td>21.04</td>
<td>1.05</td>
<td>6.92</td>
<td>64.88</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>33.44</td>
<td>1.43</td>
<td>9.36</td>
<td>44.18</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>44.84</td>
<td>3.14</td>
<td>36.52</td>
<td>25.18</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>49.39</td>
<td>3.33</td>
<td>32.96</td>
<td>17.59</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>39.18</td>
<td>2.10</td>
<td>31.00</td>
<td>34.62</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOIEE</td>
<td>31.32</td>
<td>1.94</td>
<td>9.43</td>
<td>56.62</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>39.93</td>
<td>2.78</td>
<td>11.79</td>
<td>44.69</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>54.82</td>
<td>5.21</td>
<td>42.81</td>
<td>24.08</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>59.38</td>
<td>5.48</td>
<td>43.58</td>
<td>17.81</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>44.35</td>
<td>3.64</td>
<td>24.81</td>
<td>38.58</td>
</tr>
</tbody>
</table>

RQ3: Different evolutionary search algorithms:

We compare the performance of our approach using NSGA-II against using two other different multiobjective optimization algorithms: SMS-EMOA and IBEA.

Regarding the story points, in terms of SMS-EMOA, our approach MOIEE improved between 2.38% (in TIMOB) to 18.81% (in TISTUD) in MAE, 11.11% (in MESOS) to 39.13% (in DNN) MMRE, 1.83% (in TIMOB) to 83.33% (in NEXUS)
Table 4.8: Evaluation results of our approach MOIEE in terms of story points, the state-of-the-art techniques: Linear Regression (LR), Case-Based Reasoning (CBR), and Random Forests (RF). The results for the single objective search with unlimited depth (GP-SAE) are also included discussed later in RQ4 (the best results are highlighted in bold).

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>MAE</th>
<th>MMRE</th>
<th>MdAE</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>MOIEE</td>
<td>2.02</td>
<td>0.27</td>
<td>1.57</td>
<td>57.65</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>2.72</td>
<td>0.58</td>
<td>1.56</td>
<td>43.04</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>2.91</td>
<td>0.87</td>
<td>2.46</td>
<td>38.88</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>2.81</td>
<td>0.76</td>
<td>2.38</td>
<td>40.95</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>2.76</td>
<td>0.75</td>
<td>2.14</td>
<td>42.88</td>
</tr>
<tr>
<td>DNN</td>
<td>MOIEE</td>
<td>0.41</td>
<td>0.14</td>
<td>0.31</td>
<td>78.00</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>0.80</td>
<td>0.45</td>
<td>1.00</td>
<td>57.88</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>0.78</td>
<td>0.52</td>
<td>0.63</td>
<td>58.95</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>0.58</td>
<td>0.30</td>
<td>0.42</td>
<td>69.69</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>0.57</td>
<td>0.29</td>
<td>0.38</td>
<td>70.25</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOIEE</td>
<td>0.51</td>
<td>0.32</td>
<td>0.34</td>
<td>85.44</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>1.20</td>
<td>0.54</td>
<td>1.00</td>
<td>56.93</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>1.38</td>
<td>0.66</td>
<td>0.90</td>
<td>54.74</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>1.39</td>
<td>0.51</td>
<td>0.72</td>
<td>54.72</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>0.96</td>
<td>0.42</td>
<td>0.66</td>
<td>68.74</td>
</tr>
<tr>
<td>MULE</td>
<td>MOIEE</td>
<td>1.41</td>
<td>0.41</td>
<td>0.71</td>
<td>60.66</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>2.25</td>
<td>0.95</td>
<td>1.00</td>
<td>37.08</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>2.44</td>
<td>1.15</td>
<td>1.93</td>
<td>31.70</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>2.41</td>
<td>1.12</td>
<td>1.77</td>
<td>32.74</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>2.34</td>
<td>0.97</td>
<td>1.51</td>
<td>34.53</td>
</tr>
<tr>
<td>TIMOB</td>
<td>MOIEE</td>
<td>1.23</td>
<td>0.40</td>
<td>1.07</td>
<td>74.91</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>2.06</td>
<td>0.78</td>
<td>1.12</td>
<td>58.55</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>2.10</td>
<td>0.87</td>
<td>1.95</td>
<td>58.09</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>1.97</td>
<td>0.58</td>
<td>1.51</td>
<td>60.63</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>1.81</td>
<td>0.56</td>
<td>1.27</td>
<td>63.89</td>
</tr>
<tr>
<td>TISTUD</td>
<td>MOIEE</td>
<td>0.82</td>
<td>0.23</td>
<td>0.44</td>
<td>85.01</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>1.26</td>
<td>0.37</td>
<td>0.87</td>
<td>77.19</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>1.78</td>
<td>0.44</td>
<td>1.55</td>
<td>67.83</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>1.49</td>
<td>0.49</td>
<td>1.11</td>
<td>73.10</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>1.29</td>
<td>0.38</td>
<td>1.01</td>
<td>76.72</td>
</tr>
<tr>
<td>XD</td>
<td>MOIEE</td>
<td>0.76</td>
<td>0.27</td>
<td>0.50</td>
<td>72.34</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>1.13</td>
<td>0.55</td>
<td>1.00</td>
<td>58.99</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>1.57</td>
<td>0.68</td>
<td>1.10</td>
<td>43.48</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>1.15</td>
<td>0.46</td>
<td>0.91</td>
<td>58.60</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>1.02</td>
<td>0.38</td>
<td>0.94</td>
<td>63.01</td>
</tr>
<tr>
<td>NEXUS</td>
<td>MOIEE</td>
<td>0.19</td>
<td>0.12</td>
<td>0.03</td>
<td>81.96</td>
</tr>
<tr>
<td></td>
<td>GP-SAE</td>
<td>0.33</td>
<td>0.27</td>
<td>0.12</td>
<td>70.58</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>0.48</td>
<td>0.42</td>
<td>0.36</td>
<td>57.06</td>
</tr>
<tr>
<td></td>
<td>CBR</td>
<td>0.41</td>
<td>0.31</td>
<td>0.23</td>
<td>63.69</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>0.35</td>
<td>0.29</td>
<td>0.22</td>
<td>68.87</td>
</tr>
</tbody>
</table>
4.4. Evaluation

Table 4.9: A comparison of the MOIEE with both Choetkiertikul et. al. and Porru et. al. using the Mean Absolute Error (MAE)

<table>
<thead>
<tr>
<th>Project</th>
<th>MOIEE</th>
<th>Choetkiertikul et. al.</th>
<th>Porru et. al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>2.02</td>
<td>2.67</td>
<td>5.69</td>
</tr>
<tr>
<td>DNN</td>
<td>0.41</td>
<td>0.47</td>
<td>1.08</td>
</tr>
<tr>
<td>MESOS</td>
<td>0.51</td>
<td>0.76</td>
<td>1.23</td>
</tr>
<tr>
<td>MULE</td>
<td>1.41</td>
<td>2.32</td>
<td>3.37</td>
</tr>
<tr>
<td>NEXUS</td>
<td>0.19</td>
<td>0.21</td>
<td>0.39</td>
</tr>
<tr>
<td>TIMOB</td>
<td>1.23</td>
<td>1.44</td>
<td>1.76</td>
</tr>
<tr>
<td>TISTUD</td>
<td>0.82</td>
<td>1.04</td>
<td>1.28</td>
</tr>
<tr>
<td>XD</td>
<td>0.76</td>
<td>1.00</td>
<td>1.86</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.91</td>
<td>1.27</td>
<td>2.08</td>
</tr>
</tbody>
</table>

Table 4.10: Comparison between our approach MOIEE and the state-of-the-art techniques (LR, CBR, RF, SMS-EMOA, and IBEA) in terms of elapsed time using Wilcoxon test and $A_{xy}$ effect size (in brackets)

<table>
<thead>
<tr>
<th>MOIEE vs.</th>
<th>LR</th>
<th>CBR</th>
<th>RF</th>
<th>SMS-EMOA</th>
<th>IBEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON</td>
<td>&lt;0.001 [0.76]</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.73]</td>
<td>&lt;0.001 [0.65]</td>
<td>&lt;0.001 [0.68]</td>
</tr>
<tr>
<td>HDFS</td>
<td>&lt;0.001 [0.77]</td>
<td>&lt;0.001 [0.73]</td>
<td>&lt;0.001 [0.75]</td>
<td>&lt;0.001 [0.59]</td>
<td>&lt;0.001 [0.63]</td>
</tr>
<tr>
<td>MAPREDUCE</td>
<td>&lt;0.001 [0.79]</td>
<td>&lt;0.001 [0.74]</td>
<td>&lt;0.001 [0.76]</td>
<td>&lt;0.001 [0.65]</td>
<td>&lt;0.001 [0.60]</td>
</tr>
<tr>
<td>YARN</td>
<td>&lt;0.001 [0.79]</td>
<td>&lt;0.001 [0.72]</td>
<td>&lt;0.001 [0.74]</td>
<td>&lt;0.001 [0.56]</td>
<td>&lt;0.001 [0.54]</td>
</tr>
<tr>
<td>MESOS</td>
<td>&lt;0.001 [0.75]</td>
<td>&lt;0.001 [0.71]</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.53]</td>
<td>&lt;0.001 [0.61]</td>
</tr>
</tbody>
</table>

Table 4.11: Comparison between our approach MOIEE and the state-of-the-art techniques (LR, CBR, RF, SMS-EMOA, and IBEA) in terms of story points using Wilcoxon test and $A_{xy}$ effect size (in brackets)

<table>
<thead>
<tr>
<th>MOIEE vs.</th>
<th>LR</th>
<th>CBR</th>
<th>RF</th>
<th>SMS-EMOA</th>
<th>IBEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>&lt;0.001 [0.62]</td>
<td>&lt;0.001 [0.65]</td>
<td>0.163 [0.57]</td>
<td>0.793 [0.61]</td>
<td>0.162 [0.56]</td>
</tr>
<tr>
<td>DNN</td>
<td>&lt;0.001 [0.78]</td>
<td>&lt;0.001 [0.68]</td>
<td>&lt;0.001 [0.58]</td>
<td>&lt;0.001 [0.55]</td>
<td>&lt;0.001 [0.62]</td>
</tr>
<tr>
<td>MESOS</td>
<td>&lt;0.001 [0.62]</td>
<td>0.104 [0.67]</td>
<td>&lt;0.001 [0.54]</td>
<td>&lt;0.001 [0.65]</td>
<td>&lt;0.001 [0.68]</td>
</tr>
<tr>
<td>MULE</td>
<td>&lt;0.001 [0.77]</td>
<td>&lt;0.001 [0.59]</td>
<td>&lt;0.001 [0.68]</td>
<td>&lt;0.001 [0.57]</td>
<td>&lt;0.001 [0.63]</td>
</tr>
<tr>
<td>TIMOB</td>
<td>&lt;0.001 [0.66]</td>
<td>&lt;0.001 [0.60]</td>
<td>&lt;0.001 [0.66]</td>
<td>0.627 [0.54]</td>
<td>0.024 [0.53]</td>
</tr>
<tr>
<td>TISTUD</td>
<td>&lt;0.001 [0.71]</td>
<td>0.097 [0.77]</td>
<td>&lt;0.001 [0.55]</td>
<td>0.104 [0.55]</td>
<td>&lt;0.001 [0.61]</td>
</tr>
<tr>
<td>XD</td>
<td>&lt;0.001 [0.67]</td>
<td>&lt;0.001 [0.57]</td>
<td>0.002 [0.62]</td>
<td>&lt;0.001 [0.63]</td>
<td>&lt;0.001 [0.60]</td>
</tr>
<tr>
<td>NEXUS</td>
<td>&lt;0.001 [0.78]</td>
<td>&lt;0.001 [0.74]</td>
<td>&lt;0.001 [0.68]</td>
<td>0.047 [0.53]</td>
<td>0.006 [0.59]</td>
</tr>
</tbody>
</table>

in MdAE, 1.08% (in NEXUS) to 11.22% (in MULE) in SA, and 7.04% (in TISTUD) to 32.07% (in MULE) in hypervolume. MOIEE improved over IBEA between 9.55% (in TIMOB) to 22.64% (in TISTUD) MAE, 16.32% (in MULE) to 36.36% (in DNN)
4.4. Evaluation

MMRE, 6.95% (in TIMOB) to 83.33% (in NEXUS) MdAE, 1.59% (in NEXUS) to 11.20% (in XD) SA, and 7.57% (in NEXUS) to 49.05% (in DNN) hypervolume (see Table 4.12).

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>MAE</th>
<th>MMRE</th>
<th>MdAE</th>
<th>SA</th>
<th>Hypervolume</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>MOIEE</td>
<td>2.02</td>
<td>0.27</td>
<td>1.57</td>
<td>57.65</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>2.24</td>
<td>0.38</td>
<td>1.71</td>
<td>53.04</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>2.29</td>
<td>0.39</td>
<td>1.90</td>
<td>51.86</td>
<td>0.59</td>
</tr>
<tr>
<td>DNN</td>
<td>MOIEE</td>
<td>0.41</td>
<td>0.14</td>
<td>0.31</td>
<td>78.08</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>0.48</td>
<td>0.23</td>
<td>0.35</td>
<td>74.76</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>0.50</td>
<td>0.22</td>
<td>0.41</td>
<td>73.46</td>
<td>0.53</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOIEE</td>
<td>0.51</td>
<td>0.32</td>
<td>0.34</td>
<td>85.44</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>0.61</td>
<td>0.36</td>
<td>0.39</td>
<td>78.49</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>0.64</td>
<td>0.40</td>
<td>0.47</td>
<td>77.35</td>
<td>0.57</td>
</tr>
<tr>
<td>MULE</td>
<td>MOIEE</td>
<td>1.41</td>
<td>0.41</td>
<td>0.71</td>
<td>60.66</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>1.64</td>
<td>0.57</td>
<td>1.35</td>
<td>54.54</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>1.59</td>
<td>0.49</td>
<td>1.14</td>
<td>55.84</td>
<td>0.52</td>
</tr>
<tr>
<td>TIMOB</td>
<td>MOIEE</td>
<td>1.23</td>
<td>0.40</td>
<td>1.07</td>
<td>74.91</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>1.26</td>
<td>0.46</td>
<td>1.09</td>
<td>73.08</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>1.36</td>
<td>0.52</td>
<td>1.15</td>
<td>69.29</td>
<td>0.52</td>
</tr>
<tr>
<td>TISTUD</td>
<td>MOIEE</td>
<td>0.82</td>
<td>0.23</td>
<td>0.44</td>
<td>85.01</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>1.01</td>
<td>0.32</td>
<td>0.74</td>
<td>81.74</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>1.06</td>
<td>0.31</td>
<td>0.90</td>
<td>80.8</td>
<td>0.63</td>
</tr>
<tr>
<td>XD</td>
<td>MOIEE</td>
<td>0.76</td>
<td>0.27</td>
<td>0.50</td>
<td>72.43</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>0.88</td>
<td>0.40</td>
<td>0.55</td>
<td>68.14</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>0.96</td>
<td>0.42</td>
<td>0.71</td>
<td>65.133</td>
<td>0.53</td>
</tr>
<tr>
<td>NEXUS</td>
<td>MOIEE</td>
<td>0.19</td>
<td>0.12</td>
<td>0.03</td>
<td>81.96</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>0.22</td>
<td>0.18</td>
<td>0.18</td>
<td>80.67</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>0.20</td>
<td>0.16</td>
<td>0.18</td>
<td>81.08</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Table 4.12: Evaluation results for MOIEE in terms of story points using different multi-objective optimization algorithms (SMS-EMOA:S-metric selection evolutionary multi-objective algorithm, and IBEA:Indicator-based evolutionary algorithm).

Regarding the resolution time, among those eight projects, MOIEE improved over SMS-EMOA between 6.61% (in MESOS) to 26.85% (in YARN) MAE, 5.31% (in HDFS) to 24.76% (in YARN) MMRE, 12.16% (in HDFS) to 26.85% (in YARN) MdAE, 5.75% (in MESOS) to 15.21% (in MAPREDUCE) SA, and 7.69% (in COMMON) to 15.87% (in HDFS) hypervolume. It also improved over IBEA between be-
tween 14.07% (in MESOS) to 18.83% (in HDFS) MAE, 6.38% (in MAPREDUCE) to 28.77% (in COMMON) MMRE, 19.58% (in MAPREDUCE) to 33.05% (in COMMON) MdAE, 11.18% (in MAPREDUCE) to 17.86% (in COMMON) SA, and 8.19% (in MAPREDUCE) to 28.07% (in HDFS) hypervolume is shown in Table 4.13.

The Wilcoxon test (see Table 4.10 (resolution time) and Table 4.11 (story points)) also confirms that the improvement of our approach is significant ($p < 0.001$) in all cases (resolution time) and 9/16 cases (story points) with the effect size greater than 0.53 in all projects.

Table 4.13: Evaluation results for MOIEE in terms of elapsed time using different multi-objective optimization algorithms (SMS-EMOA:S-metric selection evolutionary multi-objective algorithm, and IBEA:Indicator-based evolutionary algorithm)

<table>
<thead>
<tr>
<th>Project</th>
<th>Technique</th>
<th>MAE</th>
<th>MMRE</th>
<th>MdAE</th>
<th>SA</th>
<th>Hypervolume</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON</td>
<td>MOIEE</td>
<td>20.66</td>
<td>0.99</td>
<td>3.99</td>
<td>60.37</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>23.9</td>
<td>1.21</td>
<td>4.91</td>
<td>54.15</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>25.43</td>
<td>1.39</td>
<td>5.96</td>
<td>51.22</td>
<td>0.62</td>
</tr>
<tr>
<td>HDFS</td>
<td>MOIEE</td>
<td>15.90</td>
<td>0.89</td>
<td>2.96</td>
<td>62.48</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>18.19</td>
<td>0.94</td>
<td>3.37</td>
<td>57.08</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>19.59</td>
<td>1.21</td>
<td>3.84</td>
<td>53.77</td>
<td>0.57</td>
</tr>
<tr>
<td>MAPREDUCE</td>
<td>MOIEE</td>
<td>21.16</td>
<td>1.32</td>
<td>6.65</td>
<td>66.72</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>26.77</td>
<td>1.74</td>
<td>8.76</td>
<td>57.91</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>25.43</td>
<td>1.41</td>
<td>8.27</td>
<td>60.01</td>
<td>0.61</td>
</tr>
<tr>
<td>YARN</td>
<td>MOIEE</td>
<td>21.04</td>
<td>1.05</td>
<td>6.92</td>
<td>64.88</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>26.69</td>
<td>1.31</td>
<td>8.76</td>
<td>55.45</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>25.49</td>
<td>1.34</td>
<td>8.75</td>
<td>57.46</td>
<td>0.64</td>
</tr>
<tr>
<td>MESOS</td>
<td>MOIEE</td>
<td>31.32</td>
<td>1.94</td>
<td>9.43</td>
<td>56.62</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>SMS-EMOA</td>
<td>33.54</td>
<td>2.22</td>
<td>9.43</td>
<td>53.54</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>IBEA</td>
<td>36.45</td>
<td>2.38</td>
<td>10.11</td>
<td>49.51</td>
<td>0.57</td>
</tr>
</tbody>
</table>

When we use MAE, MMRE, MdAE, and SA as evaluation criteria, MOIEE is still the best approach, consistently outperforming SMS-EMOA and IBEA across all thirteen projects.
4.4. Evaluation

Answer to RQ3: The evaluation results demonstrate that while using NSGA-II consistently produces the best results, our proposed approach also performs well with the other alternative multiobjective algorithms: SMS-EMOA, and IBEA.

RQ4: Benefits from Multi-objective Approach

In terms of resolution time estimation, results from Table 4.7 show that the single objective approach with unlimited depth tree (GP-SAE) even outperforms all the baselines and state-of-the-art techniques. However, using tree size as the second objective has brought significant improvement in estimation accuracy. Across the five projects we studied, MOIEE achieved from 22% to 37% improvement over GP-SAE in MAE. Overall, the improvement achieved by MOIEE over GP-SAE method is 29.11% in terms of MAE, 35.69% in terms of MMRE, 30.18% in terms of MdAE, and 33.57% in terms of SA, averaging across all projects.

In terms of story points estimation, Table 4.8 shows that MOIEE improved over (GP-SAE) between 26.81% (in APSTUD) to 57.50% (in MESOS) in MAE, 37.83% (in TISTUD) to 68.88% (in DNN) in MMRE, 4.46% (in TIMOB) to 75% (in NEXUS) in MdAE, and 10.80% (in TISTUD) to 75.67% (in MULE) in SA.

In addition, the Wilcoxon test also confirmed that the improvement brought by using our multi-objective approach is significant ($p < 0.001$) in all cases with effect size from 0.54 to 0.72 in both (resolution time and story points estimation) (see the first row in Table 4.14 and 4.15).

Our approach of using tree size as the second objective is effective not only in improving the accuracy of the estimation model but also in the reducing its complexity. As can be seen in Table 4.14 and 4.15, the average tree sizes of the solution estimation models produced by MOIEE were significantly less than those produced GP-SAE (e.g. 16 nodes versus 425 nodes for the Hadoop Common project, and 22 nodes versus 1171 nodes for the APSTUD project). The approach of setting a fixed depth limit (depth
Table 4.14: Comparison between our multi and single objective (MOIEE vs. GP-SAE) using Wilcoxon test and $\hat{A}_{12}$ effect size (in brackets) in terms of elapsed time. The second row reports the average tree size (i.e. the number of nodes) of a solution estimation model – the first number produced by MOIEE while the second number produced by GP-SAE.

<table>
<thead>
<tr>
<th></th>
<th>COMMON</th>
<th>HDFS</th>
<th>MAPREDUCE</th>
<th>YARN</th>
<th>MESOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOIEE Tree size</td>
<td>&lt;0.001 [0.60]</td>
<td>&lt;0.001 [0.59]</td>
<td>&lt;0.001 [0.64]</td>
<td>&lt;0.001 [0.62]</td>
<td>&lt;0.001 [0.65]</td>
</tr>
<tr>
<td></td>
<td>16 vs. 425</td>
<td>20 vs. 353</td>
<td>14 vs. 214</td>
<td>18 vs. 437</td>
<td>24 vs. 220</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>COMMON</th>
<th>HDFS</th>
<th>MAPREDUCE</th>
<th>YARN</th>
<th>MESOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOIEE Tree size</td>
<td>&lt;0.001 [0.58]</td>
<td>&lt;0.001 [0.56]</td>
<td>&lt;0.001 [0.60]</td>
<td>&lt;0.001 [0.58]</td>
<td>&lt;0.001 [0.61]</td>
</tr>
<tr>
<td></td>
<td>16 vs. 46</td>
<td>20 vs. 43</td>
<td>14 vs. 34</td>
<td>18 vs. 39</td>
<td>24 vs. 70</td>
</tr>
</tbody>
</table>

= 10) is also not as effective as the multi-objective approach. Although using this approach reduced the tree size of the solution estimation models, it is still inferior to the multi-objective approach in terms of accuracy (see the bottom part of Table 4.14 and 4.15). These results clearly demonstrate the benefit of using our multi-objective approach in terms of producing both accurate and simple estimation models.

Answer to RQ4: Using multi-objective approach provides more accurate and robust estimates than single-objective models.

RQ5: Cross-Project Estimation

In terms of story points estimation, among those eight projects, Table 4.16 shows the performance of our MOIEE for 56 cross-project estimation experiments. More explicitly, we used the issues from APSTUD project for training (source) to obtain an estimation model and then applied this model for the issues in DNN project for test (target). In all cases, when we used our approach within-project estimation, it performed better than when we used it for cross-project estimation. Meanwhile, our approach for cross-project estimation outperformed the baseline techniques. For example, we used DNN project (source) and APSTUD project (target), our approach for cross-project indicated 24.34% reduction than within-project, while it achieved 22.60% over the median (baseline technique) in terms of MAE.
### Table 4.15: Comparison between our multi and single objective (MOIEE vs. GP-SAE) using Wilcoxon test and $A_{xy}$ effect size (in brackets) in terms of story points. The second row reports the average tree size (i.e. the number of nodes) of a solution estimation model – the first number produced by MOIEE while the second number produced by GP-SAE.

<table>
<thead>
<tr>
<th></th>
<th>APSTUD</th>
<th>DNN</th>
<th>MESOS</th>
<th>MULE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tree size</strong></td>
<td>&lt;0.001 [0.58]</td>
<td>&lt;0.001 [0.67]</td>
<td>&lt;0.001 [0.72]</td>
<td>&lt;0.001 [0.61]</td>
</tr>
<tr>
<td>GP-SAE (unlimited depth)</td>
<td>22 vs. 1171</td>
<td>17 vs. 266</td>
<td>20 vs. 221</td>
<td>14 vs. 296</td>
</tr>
<tr>
<td></td>
<td>TIMOB</td>
<td>TISTUD</td>
<td>XD</td>
<td>NEXUS</td>
</tr>
<tr>
<td>MOIEE</td>
<td>&lt;0.001 [0.65]</td>
<td>&lt;0.001 [0.59]</td>
<td>&lt;0.001 [0.54]</td>
<td>&lt;0.001 [0.60]</td>
</tr>
<tr>
<td>Tree size</td>
<td>18 vs. 545</td>
<td>23 vs. 798</td>
<td>15 vs. 554</td>
<td>12 vs. 215</td>
</tr>
<tr>
<td><strong>Tree size</strong></td>
<td>&lt;0.001 [0.61]</td>
<td>&lt;0.001 [0.66]</td>
<td>&lt;0.001 [0.68]</td>
<td>&lt;0.001 [0.62]</td>
</tr>
<tr>
<td>GP-SAE (depth = 10)</td>
<td>22 vs. 44</td>
<td>17 vs. 40</td>
<td>20 vs. 41</td>
<td>14 vs. 35</td>
</tr>
<tr>
<td></td>
<td>TIMOB</td>
<td>TISTUD</td>
<td>XD</td>
<td>NEXUS</td>
</tr>
<tr>
<td>MOIEE</td>
<td>&lt;0.001 [0.60]</td>
<td>&lt;0.001 [0.56]</td>
<td>&lt;0.001 [0.56]</td>
<td>&lt;0.001 [0.55]</td>
</tr>
<tr>
<td>Tree size</td>
<td>18 vs. 42</td>
<td>23 vs. 47</td>
<td>15 vs. 42</td>
<td>12 vs. 38</td>
</tr>
</tbody>
</table>

In terms of resolution time estimation, Table 4.17 reports the MAE produced by our approach in cross-project settings. For example, when we used the issues from Hadoop Common for training to obtain an estimation model and then applied this model for the issues in Hadoop HDFS, our approach achieved 16.20 MAE. In the within-project setting, i.e. the training was done using Hadoop HDFS, our approach achieved 15.90 MAE (see Table 4.3). The decreased performance in this case was relatively small (only $\sim 2\%$), which is also observed in the other cases. We also observe that estimations done cross the four projects in Hadoop were more accurate than those performed cross one of the four projects in Hadoop and the Apache Mesos project. This may be due to the fact that the four Hadoop projects share many commonalities than those in a totally different project like Mesos.

Hence, these results suggest that our approach can be used for cross-project estimation with a small sacrifice in accuracy.
### 4.4. Evaluation

Table 4.16: MAE produced by our approach MOIEE in cross-project settings in terms of story points, source (trained project) and target (tested project)

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>MOIEE</th>
<th>Source</th>
<th>Target</th>
<th>MOIEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>APSTUD</td>
<td>DNN</td>
<td>0.66</td>
<td>NEXUS</td>
<td>APSTUD</td>
<td>2.56</td>
</tr>
<tr>
<td></td>
<td>MESOS</td>
<td>0.88</td>
<td></td>
<td>DNN</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>MULE</td>
<td>2.13</td>
<td></td>
<td>MESOS</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>NEXUS</td>
<td>0.46</td>
<td></td>
<td>MULE</td>
<td>2.20</td>
</tr>
<tr>
<td></td>
<td>TIMOB</td>
<td>2.29</td>
<td></td>
<td>TIMOB</td>
<td>2.15</td>
</tr>
<tr>
<td></td>
<td>TISTUD</td>
<td>1.20</td>
<td></td>
<td>TISTUD</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td>XD</td>
<td>1.13</td>
<td></td>
<td>XD</td>
<td>1.07</td>
</tr>
<tr>
<td>DNN</td>
<td>APSTUD</td>
<td>2.67</td>
<td>TIMOB</td>
<td>APSTUD</td>
<td>2.17</td>
</tr>
<tr>
<td></td>
<td>MESOS</td>
<td>0.61</td>
<td></td>
<td>DNN</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>MULE</td>
<td>1.48</td>
<td></td>
<td>MESOS</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>NEXUS</td>
<td>0.33</td>
<td></td>
<td>MULE</td>
<td>2.27</td>
</tr>
<tr>
<td></td>
<td>TIMOB</td>
<td>2.00</td>
<td></td>
<td>NEXUS</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>TISTUD</td>
<td>1.77</td>
<td></td>
<td>TISTUD</td>
<td>1.09</td>
</tr>
<tr>
<td></td>
<td>XD</td>
<td>1.15</td>
<td></td>
<td>XD</td>
<td>1.31</td>
</tr>
<tr>
<td>MESOS</td>
<td>APSTUD</td>
<td>2.35</td>
<td>TISTUD</td>
<td>APSTUD</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>0.62</td>
<td></td>
<td>DNN</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>MULE</td>
<td>2.05</td>
<td></td>
<td>MESOS</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>NEXUS</td>
<td>0.3</td>
<td></td>
<td>MULE</td>
<td>2.19</td>
</tr>
<tr>
<td></td>
<td>TIMOB</td>
<td>1.92</td>
<td></td>
<td>NEXUS</td>
<td>0.40</td>
</tr>
<tr>
<td></td>
<td>TISTUD</td>
<td>1.40</td>
<td></td>
<td>TIMOB</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>XD</td>
<td>1.08</td>
<td></td>
<td>XD</td>
<td>0.96</td>
</tr>
<tr>
<td>MULE</td>
<td>APSTUD</td>
<td>2.14</td>
<td>XD</td>
<td>APSTUD</td>
<td>2.45</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>0.62</td>
<td></td>
<td>DNN</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>MESOS</td>
<td>0.59</td>
<td></td>
<td>MESOS</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>NEXUS</td>
<td>0.29</td>
<td></td>
<td>MULE</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>TIMOB</td>
<td>1.96</td>
<td></td>
<td>NEXUS</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>TISTUD</td>
<td>1.39</td>
<td></td>
<td>TIMOB</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td>XD</td>
<td>0.94</td>
<td></td>
<td>TISTUD</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 4.17: MAE produced by our approach MOIEE in cross-project settings, trained using a project in the row and tested on a project in the column.

<table>
<thead>
<tr>
<th>Project</th>
<th>COMMON</th>
<th>HDFS</th>
<th>MAPREDUCE</th>
<th>YARN</th>
<th>MESOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON</td>
<td></td>
<td>16.20</td>
<td>25.57</td>
<td>25.83</td>
<td>35.85</td>
</tr>
<tr>
<td>HDFS</td>
<td>25.07</td>
<td></td>
<td>30.93</td>
<td>31.64</td>
<td>36.96</td>
</tr>
<tr>
<td>MAPREDUCE</td>
<td>24.74</td>
<td>19.75</td>
<td></td>
<td>29.93</td>
<td>34.84</td>
</tr>
<tr>
<td>YARN</td>
<td>27.90</td>
<td>20.10</td>
<td>30.72</td>
<td></td>
<td>35.69</td>
</tr>
<tr>
<td>MESOS</td>
<td>25.65</td>
<td>20.15</td>
<td>30.06</td>
<td>29.75</td>
<td></td>
</tr>
</tbody>
</table>
4.4. Evaluation

Answer to RQ5: Although our proposed approach can be used for cross-project estimation, it is a more effective for within-project estimation.

4.4.4 Threats to validity

To mitigate threats to construct validity, we used real world data from issues reported in large open source projects. We collected the common issue features, the actual time and actual story points took to resolve issues. In terms of the conclusion validity, we carefully selected unbiased error measures and applied a number of statistical tests to verify our assumptions \(^{329}\). Our study was performed on two different datasets with variable project’s sizes. Furthermore, we carefully followed recent best practices in evaluating effort estimation models (e.g. \(^{327}\)) to minimize conclusion instability.

Another threat is related to the random initialization of the first generation. Therefore, a single run of an experimental study may deliver results that can be affected either by the favorable initial random selection or the bad randomly selected point \(^{332}\). To avoid this problem, we have conducted a multiple run (30 runs in this study) and chose the median result.

We strictly followed Porru et. al.’s method \(^{338}\), however, we have re-applied our own version as the original implementation of this method was not published. Therefore, in our implementation, we acknowledge that we might not reflect all the details of Porru et. al.’s approach. To overcome this threat, we have examined our multi-objective approach by using the dataset provided in Porru et. al.’s work. Our approach results were consistent with Porru et. al.’s approach results.

To overcome the external validity threat, we have considered 12,938 issues from thirteen different projects. These issues are significantly diverse in size, a team of developers, complexity, and community. By this way, different contexts can be char-
acterised by some specific projects and human factors (e.g. team structure and communication, time, team effort and other constraints, and so on). However, we cannot claim that our datasets are representative of all kind of software projects, and that our results can generalize to all software projects, especially those in commercial settings.

4.5 Related work

In a software project, effort estimation is the process of estimating the necessary effort such as scheduling, and allocating resources, to complete different software tasks in software project management and meet delivery deadlines \cite{377}. This process aims to achieve a correct, realistic and reliable project planning. The correctness of this process is one of the key factors for a successful software project \cite{2,378,380}. Predicting resolution time and story points could be considered as a form of measure for effort in software projects. Effort estimation is a core issue for project managers as well as to the development teams in order to schedule a plan of costing and timing for the future releases. For example, the project managers may need to estimate the time it would take for completing a project task, on the other hand, the development teams need to estimate the specified effort for completing a development task.

In past decades, effort estimation research, in general, were divided into expert-based methods (i.e., judgments are done by human expertise), and model-based method where the new projects’ predictions are made by using data from old projects \cite{379,381}. The approaches in most of the existing effort estimation work \cite{14,301,382,390} were like a waterfall software development model. These approaches rely on a set of manually designed characterizing features. They were applied in effort estimation for developing a complete software system.

An extensive number of studies have been presented in the field of effort estimation (e.g. \cite{307,391,392}). The intention of these work was to provide a reliable
4.5. Related work

Effort estimation methods. Over the years, whether adopting theoretical models, well-known formal models or analogy-base models, researchers have tried to build an efficient model for effort estimation. Nevertheless, no agreements were reached regarding the most reliable used approach. Several studies have worked on building effort estimation models using variable data analytic techniques such as neural networks (e.g. [394,395]), Bayesian network (e.g. [396,397]), fuzzy (e.g. [398,399]) automatically transformed linear model (ATLM) (e.g. [400]), Deep Neural Networks (e.g. [401]), regression (e.g. [402]). All mentioned studies have built their computational intelligence models based on data from a range of past projects. Hence, these models are suitable only for a specific kind of project. While most existing work focuses on effort estimation of a whole project, none of them have been done at issue level. Besides, some of these studies have used the machine learning technique, others have used a genetic algorithm, but none of the conducted studies have adopted the multi-objective evolutionary algorithms.

There is an emerging interest in predicting the fixing time of a bug, which was initiated by the work of [194]. These work (e.g. [403–406]) use machine learning techniques (e.g. kNN in [194] or Random Forests in [339,347,407]) to build their prediction models. For example, the work in [194] estimates the fixing time of a bug by finding the previous bugs that have similar description to the given bug (using text similar techniques) and using the known time of fixing those previous bugs. Using decision trees and other machine learning techniques, the work in [408] predict the lifetime of Eclipse bugs based on several primitive features of a bug such as severity, component, and number of comments. The work in [347] explored a different set of issue features including location, reporter and description. The time when the prediction is made also affect the predictive performance as shown in the study in [346]. They tested the predictive models with initial bug report data as well as those with post-
submission information and found that inclusion of post-submission bug report data of up to one month can further improve prediction models. Most of those techniques used classifiers which do not deal with continuous response variables, they need to discretize the fix-time into categories, e.g. within 1 month, 1 year and more than 1 year as in [347]. This is one of the key difference to our work since we are able to predict the exact resolution time. In addition, our work offers an alternative in which we propose a search-based evolutionary approach to the problem. The work proposed a multi-objective search-based approach to estimate issue resolution time. Their approach leverages evolutionary algorithms to find robust estimation models. The search is guided simultaneously by two contrasting objectives: maximizing the accuracy of an estimation model and minimizing the complexity of the estimation model and hence helps produce accurate and simple estimation models. The obtained results revealed that the proposed approach outperformed other commonly used estimation models (linear regression, case-based reasoning, and random forests).

In the agile context, several works have used machine learning to estimate story points [307,338]. These studies investigated the performance of different techniques (e.g., Neural Networks [409]) with respect to classification task in addition to the performance of some other traditional machine learning algorithms (e.g., Decision Trees, K-NN, Naive Bayes, and SVMs) [307]. Porru et. al. [338] estimated story points by using machine learning classifier. From eight open source projects, the authors extracted their attributes (textual features, issue type fields, and components) from issue reports. They found that those attributes have proven that they are highly project-dependent features which are crucial to estimating story point. Scott et al. [340] also employed machine learning classifier to assign story points to issue reports. The authors built a predictive model that use developers’ features. They then compared the performance of their model with other models that used features extracted from the text of issues.
They found that the model that based on developers’ features outperforms models using text features.

Abrahamsson et. al. [409] used regression models and neural networks to develop an effort prediction model for iterative setting in software development. In this work, the model has been built after each iteration to estimate effort for the next iteration, and that what makes it different from other traditional effort estimation models which are built at the end of the project. Later on, [410] worked on estimating story points. For this purpose, the author built a classifier and used it on the user stories which were provided by an agile company. They achieved their best results by using the SVM algorithm. Haugen et. al. [360] compared the estimated story points during release planning. The focus of the study was on the estimation of user stories that have been mediated by developers during the planning poker sessions (i.e., human side judgments). To assess estimation ability of the team, the author compared the story points at the initial release planning meeting with assigned story points once the task was implemented. The results indicated that this estimation process (i.e., planning poker) improved the estimation performance of the development team.

Recently, Choetkiertikul et. al. [307] applied deep learning techniques to deal with effort estimation problems at user story level (i.e., issue). To solve this problem, the authors leveraged both long short-term memory (LSTM) and recurrent highway network to build a prediction model for estimating story points. The features of this model were learned from the description, title, and comments related to an issue report. The results showed a significant improvement of LSTM over the baseline method. Machine learning approaches such as those in by Porru et. al. and Choetkiertikul et. al. however offer “black box” estimation models, which are limited in explaining their predictions.

In the area of search-based software engineering, substantial works have been
Most of the recent work in the context of effort prediction can be found in the review paper of [41]. While most of the existing work in this space use single-objective search, a few of them (e.g. [14, 15]) has recently proposed multi-objective search approach to effort estimation. For example, a recent study done by [14] employed NSGA-II with two objectives sum of absolute error and confidence interval. Sarro et. al. [14] built the effort estimation models using multi-objective approach for the whole projects. The prediction process considered both predictive accuracy and predictive confidence. The results showed the multi-objective method outperformed another state of the art in effort estimation.

There are however several key differences from our work and Sarro et. al.’s work. First, we built effort estimation models in terms of both resolution time and story point (effort) for a single software issue rather than for the whole project (as done by Sarro et. al.). This makes our work more relevant and applicable to the modern agile software development settings where the focus is at the issue level. Second, we used the tree size as the second objective function to simultaneously manage the parsimonious and accuracy of generated estimation models. Hence, our approach does not impose any fixed structure or depth on the candidate models, which is different from Sarro et. al.’s approach. In addition, while Sarro et. al. used only three mathematical operators, we used a wider range of thirteen mathematical operators and thus accommodate a larger search space.

4.6 Chapter summary

In this chapter, we have presented a multi-objective search-based approach to predict both resolution time and effort in story points at an issue level in software projects. Our approach leverages evolutionary algorithms to find robust estimation models. The
search is guided simultaneously by two contrasting objectives: maximizing the accuracy of an estimation model and minimizing the complexity of the estimation model. A comprehensive evaluation on thirteen software projects with 12,937 issues demonstrated that our approach consistently outperforms not only common naive baselines but also state-of-the-art techniques in all datasets. Our results also demonstrate the benefit of using the complexity measure as the second fitness function since this approach produced more accurate but less complex estimation model than the single-objective approach did. Results from our cross-project experiments also suggest that our approach is also applicable for cross-project estimation. In the next chapter, we present an approach to recommend developers (patch author) with a list of suitable reviewers for code changes made for resolving issues.
Chapter 5

Workload-Aware Code Reviewer

Recommendation

Code is one of the important and efficient means in the software development process. In the software quality assurance process, code review has been recognized as an essential stage to early identify defects in a set of code changes (i.e. a patch), aiming to control quality in software projects \[18,19\]. In this code review process, developers (patch authors) submit their patch to be reviewed by other developers (code reviewers) before being integrated into the main repositories \[17\].

In recent years, a lightweight variant called Modern Code Review (MCR), a less formal variant of code reviews that limits the inefficiencies of inspections, has been incorporated to facilitate the code review process \[412\] in modern software development. Several open source software and industrial projects use an MCR process. A dedicated code review tool (e.g., web-based tools such as Gerrit\[1\]) is used to perform code reviews within those projects. The code review process in those tools generally begins with a patch author inviting a set of code reviewers to review the newly-submitted patch. Then, reviewers examine code changes in the patch and identify weaknesses. When

\[1\]https://www.gerritcodereview.com
one or more code reviewers agree that the patch is of sufficient quality, the patch
will be integrated into main software repositories. The MCR practice mainly empha-
sizes on team member collaboration, aiming to attain and maintain a high quality of
software products.

The efficiency and effectiveness of code review is associated with several factors
such as experience and knowledge of reviewers. For example, prior work shows that
reviewers with in-depth knowledge in terms of code and context provide constructive
feedback \[219, 220\]. Typically, in software projects, reviewer expertise related to the
changed files in a patch and prior review collaboration are the main factors that a patch
author considers when inviting reviewers \[25, 220\]. On the other hand, patches reviewed
by a code reviewer who has little past involvement in the changed files are more likely
to suffer from an ineffective code review (e.g., poor review discussions) \[413\].

However, the review participation is likely to be low in the lightweight MCR
process due to its informal nature. In other words, not all the invited reviewers will
respond to a review invitation \[23\]. The low number of participated reviewers also
have a negative impact on software quality and reviewing timeliness. For example,
Mcintosh et al. \[222\] and Thongtanumam et al. \[21\] found that a low level of review
participation can have a negative impact on software quality. Kononenko et al. \[20\]
also found that the number of involved reviewers has an effect on the quality of the
code review process. Hence, reviewer participation is regarded as the main challenge
in MCR processes \[23, 223, 224\]. Moreover, there is a significant amount of human
effort involved, in addition to the challenge of understanding the defect patterns and
the suitable code to check that pattern \[220\].

Selecting the right reviewers is one of the main challenges that patch authors can
face in the code review process, particularly, when the process is manually performed.
There is strong empirical evidence from the open source software (OSS) domain on
the significance of identifying the most suitable reviewers to maintain high-quality code review process [17,414]. Hence, it is valuable for the developers to have an efficient reviewer recommendation approach to support their decision regarding reviewer selection.

With an ultimate goal of increasing review effectiveness and participation, several studies have proposed an approach to identify code reviewers [21,22,24,220]. However, many of these studies formulated the peer reviewers recommendation problem where the candidate reviewers are ranked and recommended only based on their expertise and past involvement. Recently, Ouni et al. [22] proposed a reviewer recommendation approach that uses both reviewer expertise and the past collaboration.

Other important factors such as the workload of code reviewers should be considered when selecting a reviewer to increase the effectiveness hence the quality of code review. It is possible that code reviewers who have a large number of review tasks may not respond to the author invitation. Ruangwan et al. [23] find that the number of review invitations can have an impact on the participation decision of a reviewer. Furthermore, the rate of code review can highly impact on reviewers’ performance and hence influence their reviewing quality. Experienced reviewers who have a high workload are more likely to not approve a patch than reviewers with fewer reviews [25–27].

In this paper, we aim to fill that gap. We propose a Workload-aware Reviewer Recommendation approach called WLRRec. We employ a multi-objective search-based evolutionary approach to find the most appropriate reviewers. Specifically, we leverage a meta-heuristic technique, namely genetic algorithms (GA), to generate a large number of candidate selection of reviewers and search for the ones that are optimal with respect to a number of objectives.

Our current work explores two objectives which guide our search algorithms.
The first objective is to maximize the reviewers expertise to achieve effective code review. It is possible that reviewers might be possibly burdened with a big number of review tasks. This would potentially increase the number of awaiting reviews which disadvantages the efficiency of the review process \[482\]. Hence, our second objective is to minimize the workload difference between developers, balancing the workload across all the reviewers.

The evaluation demonstrates that our search-based approach outperforms the common baselines. We also demonstrate the effectiveness of using a multi-objective approach against the single objective approach. The evaluation was performed against a dataset of 230,090 patches (which consist of 7,431 reviewers in total) we collected from four different open-source projects (Android, LibreOffice, Qt, and OpenStack). Following common standards, we use precision, recall, F-measure, and hypervolume to evaluate the performance of our models, and also use a non-parametric Wilcoxon test \[435\] and Vargha and Delaney’s statistic \[310\] to demonstrate both the statistical significance and the effect size of the results.

The remainder of this chapter is organized as follows. Section 5.1 briefly describes the modern code review process. Section 5.2 describes our multi-objective approach to solve this problem using evolutionary algorithms. Section 5.3 reports on the experimental evaluation of our approach. Related work is discussed in Section 5.4 before we conclude and outline future work in Section 5.5.

## 5.1 Modern Code Review

Recently, a tool-based Modern Code Review (MCR) process has been widely-adopted by a large number of open-source software projects (e.g., Android, Qt, LibreOffice, OpenStack). One of the main objectives of MCR is to examine patches that are submitted by a patch author where a reviewer verifies the correctness and the quality
of the submitted patches before integrating them into a main code repository [475].

Generally speaking, the review process is composed of five main steps:

1. An author uploads a patch (i.e., a set of proposed changes) to a code review tool.

2. An author invites a set of reviewers to critique the patch.

3. The invited reviewers examine the patch, and provide feedback and review scores. The review scores range from +2 to -2. A review score of +2 indicates that the patch is ready to be merged and integrated into the main code repository, which will be marked as “Merged”, while a review score of +1 indicates that the patch satisfies the reviewer’s criteria but the patch needs an additional confirmation from other reviewers. On the other hand, a review score of -1 indicates that a patch needs to be revised before an integration into the main code repository, while a review score of -2 indicates that the patch requires a major revision, which will be marked as “Abandoned”.

4. The author revises the patch to address the feedback from the reviewers and uploads a new revision.

5. When the updated patch meets the requirements of the reviewers, the reviewers mark the patch as accept or reject for integration into the main code repository of a software project.

Figure 5.1 illustrates an example review of the patch #41902 for the LibreOffice project, which was uploaded on the 31st of August 2017. In this patch, a patch author (i.e., Justin) submits a change to the Gerrit code review tool. Then, the patch author (i.e., Justin) invites a set of reviewers to critique the patch. The invited reviewers then

[https://gerrit.libreoffice.org/#/c/41902/](https://gerrit.libreoffice.org/#/c/41902/)
5.1. Modern Code Review

Figure 5.1: Motivation example from LibreOffice project

decide whether or not to accept the invitation. For this patch, the invited reviewers (i.e., Miklos and Szymon) accept the invitation, inspect the patch, provide comments, feedback, and review scores in order to make a decision whether the proposed change should be accepted and be integrated into the main code repository of the LibreOffice project.
5.2 Search-based software reviewer recommendation

This section describes our approach that uses a search-based software engineering technique to recommend a list of appropriate reviewers for a newly-submitted patch. Below, we describe the framework of our approach, evolutionary search, the solution representation, the fitness function, and an approach to select a solution from a Pareto front.

5.2.1 Approach Framework

Figure 5.2 provides an overview of our Workload-aware Reviewer Recommendation approach (WLRRec) which uses a search-based software engineering (SBSE) approach. The input of our WLRRec is a newly-submitted patch. Then, we obtain a list of reviewer candidates from the past reviews of patches that were merged or abandoned before the creation date of the newly-submitted patch. For each reviewer candidate, our approach computes five metrics. Four of the five metrics measure the experience of reviewers, i.e., (1) code authoring experience, (2) reviewing experience, (3) familiarity between the reviewer candidate and the patch author of the newly-submitted patch, and (4) reviewer participation rate in the MCR process. The another metric measures the current reviewing workload of a reviewer candidate, i.e., the number of open reviews that invite the reviewer candidate. We then use these metrics to apply the multi-objective evolutionary approach to find a set of reviewers with maximal experience and minimal workload (i.e. the most appropriate reviewers) for the newly-submitted patch.

We employs evolutionary techniques to find optimal solutions (i.e., a set of reviewer candidates) that satisfies each objective simultaneously. In this work, the search
5.2. Search-based software reviewer recommendation

is guided by two objectives: (1) maximizing the reviewer experience and (2) minimizing the reviewer workload. We employed non-dominated sorting genetic algorithm (NSGA-II) \cite{127} which is based on the principle that a population of candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution has a number of properties (i.e. chromosomes or genotype) which can be mutated and altered to derive new candidate solutions. At the end of the search process, the algorithm returns a set of feasible solutions, each of which represents a list of reviewers that satisfies our objectives. Finally, from the set of feasible solutions, we use a Pareto front to identify the optimal solution that is used for a reviewer recommendation.

Figure 5.2: An overview of our approach

5.2.2 Evolutionary search

As for a search method, we employed a multi-objective metaheuristic algorithm, namely non-dominated sorting genetic algorithm (NSGA-II) \cite{127}.

NSGA-II starts to randomly generate an initial population $P_0$ in which each individual in the population is a candidate set of reviewers. The fitness values of each individual with respect to each of two fitness functions (see Section 5.2.5) are computed. The population is then undergone a selection process. Selected individuals form
the parent to generate a new generation of individuals (i.e. offspring $Q_0$) through the
crossover and mutation operators. These genetic operators act directly on the repre-
sentation of candidate solutions (i.e. bit strings – refer to Section 5.2.3) to form
new valid representations. The mutation operator randomly chooses certain bits in
the string, and set them to opposite values (i.e. inverted from 0 to 1 and vice versa).
The crossover operators involve two parent bit strings (representing two candidate so-
lutions). A crossover point on both parents’ strings is chosen. The parts beyond that
point in both parents’ strings are then swapped to generate the offspring strings.

![Figure 5.3: An example of non-dominated fronts](image)

We search for solutions that meet both objectives: maximize the reviewer ex-
pertise which patch author needs to understand in order to avoid failure of reviewer
assignment closely and minimize the workload difference between developers. These
solutions form a Pareto front of the reviewer’s expertise and the reviewer’s workload
balance. A solution (i.e. recommended reviewer) on a Pareto front does not dominate
another solution on the same front, i.e. the former is better than the latter with re-
spect to at least one objective (e.g. reviewer expertise), and not worse in the other
objective (e.g. review workload). For example, in Figure 5.3 recommended reviewer
$R_1$ does not dominate $R_2$ since the former is lower than the latter in terms of the re-
viewer workload objective but has greater reviewer expertise. On the other hand, $R_1$
dominates $R_4$ since the $R_1$ has lower reviewer workload than $R_4$ and also has smaller
reviewer expertise.

At each generation, NSGA-II sorts the current population into a number of non-
dominated fronts (e.g. fronts 1, 2 and 3 in Figure 5.3).

In the final generation, NSGA-II returns a set of non-dominated solutions. This
evolution process continues until a fixed number of generations has been reached. The
NSGA-II is discussed in more details in the background chapter, Section 2.1.3.1.

5.2.3 Solution representation

We use a bit string of which length is the number of all reviewer candidates to represent
each candidate solution (i.e. a subset of reviewer candidates). Each bit in the string
has the value of either 0 or 1. A bit value of 1 indicates that the corresponding reviewer
is selected, while 0 indicates that the reviewer is excluded. For example, our approach
obtain eight reviewers from past reviews (i.e. Miklos, Andre, Jenkins, Justin, Robot,
Bailey, Szymon, and Johan). Then, a solution can be represented in a bit string as
shown in Figure 5.4. The bit string of 10110010 indicates that Miklos, Jenkins, Justin,
and Szymon are selected in this solution.

Since we obtain all reviewer candidates from the past reviews, there is a likely case
where we obtain a large number of reviewer candidates resulting in a long candidate
solution. Due to the computation intensive of the NSGA-II approach, it is not feasible
to promptly find optimal solutions for long candidate solutions. Hence, we shorten our
5.2. Search-based software reviewer recommendation

A solution of Reviewers (R) for a Patch (P)

<table>
<thead>
<tr>
<th></th>
<th>Miklos</th>
<th>Andre</th>
<th>Jenkins</th>
<th>Justin</th>
<th>Robot</th>
<th>Bailey</th>
<th>Szymon</th>
<th>Johan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Obtained a solution of selected reviewers after the "0" value has been rejected

Figure 5.4: An example of how a candidate solution is represented using a bit string

candidate solutions by removing reviewer candidates who have at least three metrics with zero values. The intuition behinds this heuristic is that the more metrics with zero values, the lower signal to review the patch the reviewer candidate has. Note that we have experimented all possible metric thresholds, i.e., removing reviewer candidates who have at least 1, 2, 3, 4, or 5 metrics with zero values out. We found that the metric threshold of 2 provides a reasonable length of candidate solutions (i.e., a median length of 23 - 381 reviewer candidates for a candidate solution), while it has a minimal impact on the ground-truth data (i.e., an average recall values of 0.91-0.99).

5.2.4 Reviewer Metrics

We use five metrics to measure experience and workload of reviewer candidates. These metrics will be used our fitness functions in the multi-objective evolutionary approach to find optimal solutions. We describe the calculations of our metrics.

Reviewer Code Authoring Experience (RCAE): This metric measures the proportion of past reviews that had been authored by a reviewer candidate. Intuitively, the more code authoring experience of the reviewer, the more likely that the reviewer has knowledge to review code in the patch [23]. We use an approach of [489] to measure this metric.

Reviewer Reviewing Experience (RRE): This metrics measure the proportion
of past reviews that had been reviewed by a reviewer candidate. Similarly, it is possible that a reviewer will review a patch that the reviewer has related reviewing experience. We use an approach of to measure this metric [490].

**Familiarity between the Invited Reviewer and the Patch Author (FIRPA):**
This metric measures the number of past reviews had been done by a reviewer candidate for a patch author. Prior work reports that the reviewers tend to review patches of a developer that they know previously [487]. We use an approach of [23] to measure this metric.

**Review Participation Rate (RPR):** This metric measures the proportion of participated reviews to the number of the received review invitations (i.e., the reviews that invite the reviewer candidate but s/he did not provide comments or votes to those reviews). This measure indicates that the higher the rate of the review participation, the more active the reviewer in the system] Thereby, a reviewer who assigned a high review participation rate in the system is the reviewer with more chance to respond to the submitted patch. We use an approach of to measure this metric [23].

**Number of Remaining Reviews (NRR):** This metric counts the number of open reviews where the reviewer candidate was invited but yet did not participate at the creation time of the studied patch. For example, A and B are reviewer candidates of the studied patch #1. At the time when the studied patch is created, reviewer A participated in reviewing patch #2, and #3 by providing a comment or a vote while reviewer B has participated in only patch #2. Hence, reviewer A has no remaining review while reviewer B has one remaining review.
5.2.5 Fitness functions

The search aims to optimize two objective functions simultaneously. In this section, we discuss these objective functions in details.

5.2.5.1 Maximize reviewer expertise (RevExp)

Reviewers should be familiar and have in-depth knowledge and experience with the code being reviewed. The level of expertise is essential to determine the defect proneness of software products. In the scope of MCR, both reviewer and patch author expertise are fundamental to decrease the likelihood of future patch defects. Thereby, selecting most appropriate reviewers is crucial to sustaining the effectiveness and efficiency of the code review process. Talking about reviewer expertise should include the past reviewer’s experiences (i.e. successes and failures) as this is an important indicator in shaping this expertise concerning reviewing task [24]. Reviewer expertise can be considered as a fundamental issue to determine the review participation (i.e. reviewers who have a good understanding and familiarity with the context of reviewed patches will have a better participation rate). The reviewer expertise can be quantified in terms of four dimensions which measure the experience that an invited reviewer has on a patch.

First, the Reviewer Code Authoring Experience (RCAE), this dimension measures the number of prior patches that had been authored by an invited reviewer. This code authoring experience could increase the likelihood of reviewer respond to a review invitation of a patch that they have related authoring experience. Second, Reviewer Reviewing Experience (RRE) this dimension counts the number of the prior patches that had been reviewed by an invited reviewer. The invited reviewer prefers to participate in reviewing a patch that they are familiar with (i.e. related reviewing experience). Third, Familiarity between the Invited Reviewer and the Patch Author
(FIRPA) measures how many prior patches had been reviewed by an invited reviewer for a patch author. In this context, the reviewers accept the invitation of the patch author that they know previously. The fourth dimension is to measure the Review Participation Rate (RPR) by calculating the proportion of a number of the responded review invitations to the number of the received review invitations. This measure indicates that the higher the rate of the review participation, the more active the reviewer in the system. Thereby, a reviewer who assigned a high review participation rate in the system is the reviewer with more chance to respond to the submitted patch.

Hence, the first objective is to maximize the reviewer expertise to support developers (i.e. patch authors) for selecting the most experienced reviewers aiming for high-quality software development. This fitness function is defined as follows.

\[
RevExp(S_i, p) = \sum_{r \in R} S_i(r) \left( \alpha_1 RCAE(r, p) + \alpha_2 RRE(r, p) + \alpha_3 FIRPA(r, p) + \alpha_4 RPR(r, p) \right) 
\]  

(5.1)

Where \( \alpha \) is a coefficient, \( \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \). \( R \) is a set of existing candidate reviewers where \( r \) belongs to \( R \). \( RevExp(S_i, p) \) is Reviewer Expertise where \( S \) represents a solution, \( i = 1 \) to \( R \), and \( p \) is a patch. Given reviewer \( r \) of a patch \( p \), \( RCAE(r, p) \) is Reviewer Code Authoring Experience, \( RRE(r,p) \) is Reviewer Reviewing Experience, \( FIRPA(r, p) \) is the familiarity between the Invited Reviewer and the Patch Author, and \( RPR(r, p) \) is Review Participation Rate.

### 5.2.5.2 Minimize reviewer workload (RevWl)

Review workload is an essential human factor to be considered in addition to technical and experience factors. This factor imposes a real threat on the participation rate of reviewers (i.e. a reviewer with a large number of review tasks may not have
time to review a new patch). Hence, a large review workload can lead to slow and delay reviewers feedback and increase the number of awaiting review tasks. Therefore, workload balancing for developers (patch reviewers) can increase their review participation rate and decrease the awaiting reviews. This aims to avoid unwanted delays to deliverables. Several studies \[414, 478\] also argue that the involvement of more than one reviewer in the reviewing task for a patch would minimize the reviewing workload and maximize the number of defects that found during the review.

We consider the reviewer workload as a second objective to be minimized by using Shannon entropy \[484\]. For each reviewer, the Number of Remaining Reviews (NRR) determine the total number of patches at the time where a reviewer was invited but yet did not participate at the creation time of the studied patch. For example, A and B are reviewers of the studied patch #1. At the time when the studied patch is created, reviewer A participated in reviewing patch #2, and #3 while reviewer B has not participated in reviewing those two patches yet. Hence, reviewer A has no remaining review while reviewer B has two remaining reviews. For each solution, we calculate the new_workload \((n_{-}wl)\) based on the number of the remaining reviews. For example, Figure 5.4 in Section 5.2.3 shows a bit string representation of a solution where the reviewers (Miklos, Jenkins, Justin, and Szymon) are represented with a value of 1 in this string. That means these reviewers are the selected reviewers. The number of remaining reviews (NRR) of these selected reviewers will increase by a value of one. While for reviewers who have represented a value of 0 (i.e. not selected), the NRR remains the same (i.e. no increase in the value). Formally, the new_workload \((n_{-}wl)\) is defined as follows.

\[
n_{-}wl(r, s_i) = NRR(r) + S_i(r) \quad \text{(5.2)}
\]

\(R\) is a set of existing reviewers where \(r\) belongs to \(R\). \(S\) represents a solution.
5.2. Search-based software reviewer recommendation

\( N_{\text{wl}}(r, s_i) \) is new workload where \( S \) represents a solution, \( i = 1 \) to \( R \), and \( P \) is a patch.

Prior work [413, 477] applied Shannon entropy for a code change. For balance workload, we apply Shannon entropy to measure the entropy change (i.e. the distribution of workload). We calculate the second objective using the following equation.

\[
RevWL(S_i, p) = \frac{1}{\log_2 |R|} \sum_{r \in R} \frac{n_{\text{wl}}(r, s_i)}{\sum_{r \in R} n_{\text{wl}}(r, s_i)} \times \log_2 \left( \frac{n_{\text{wl}}(r, s_i)}{\sum_{r \in R} n_{\text{wl}}(r, s_i)} \right)
\]  

(5.3)

\( R \) is a set of existing reviewers, and reviewer \( r \) belongs to \( R \). \( RevWL(S_i, p) \) is Reviewer Workload where \( S \) represents a solution, \( i = 1 \) to \( R \), and \( P \) is a patch.

5.2.6 Selecting a solution from a Pareto front

Our approach the NSGA-II returns a set of feasible solutions namely Pareto front solutions that represent the optimal trade-off between the reviewer expertise and the reviewer workload. Choosing which one of these solutions to use is often a user-specific decision. Researchers have proposed various approaches to help select a single solution from a Pareto front (e.g. knee points [312] or the best point (corner) [257]). In particular, the knee point approach has widely been used in previous work (e.g. [313–316]). This approach measures the Euclidean distance of each solution on the Pareto front from the reference point. This reference point has the optimal values for each objective function. The solution we select (denoting as \( ROS \)) is the one closest to the reference point (i.e. minimizing the distance). A depict of the Pareto front and knee point is graphically interpreted in Figure 5.4. The Pareto front is composed of eight non-dominated solutions (i.e. reviewers): \( R_1 \) and \( R_8 \) are the corner Pareto front
solutions, the other solutions indicate the intermediate trade-offs. Point $R_4$ reflects the knee point where we can attain the lowest loss in both objectives (i.e. to avoid the deterioration of the reviewer expertise which can be as a result of the marginal improvement of the reviewer workload and vice versa).

![Graphical representation for the Pareto front and knee point](image)

**Figure 5.5:** A graphical representation for the Pareto front and knee point

Formally, given a Pareto front $P = (R_1, R_2, R_3, ..., R_P)$ where ($R$ reviewers including knee point $R_K$) $\in P$, and assume that the reference point has the maximize the reviewer expertise ($\text{REXP}_{\text{max}}$) and minimize the workload difference between developers ($\text{RWL}_{\text{max}}$), we calculate the recommended optimal solution ($\text{ROS}$) using the following formula:

$$\text{ROS} = \sqrt{(\text{REXP}_{\text{max}} - \text{REXP}(R_i))^2 + (\text{RWL}_{\text{max}} - \text{RWL}(R_i))^2} \quad (5.4)$$

where $R_i \in P$. In our evaluation, we selected a solution from the Pareto fronts returned
5.3. Evaluation

This section discusses the evaluation that we have carried out for our approach. We first describe how data is collected and preprocessed for our study. Next, we describe the experimental settings, discuss the performance measures, and report our results. Our empirical evaluation aims to answer the following research questions:

- **RQ1. Sanity Check:** *Is the multi-objective search-based approach suitable for providing a more accurate reviewer recommendation for code changes?*
  
  This sanity check requires us to compare our multi-objective approach against the a common naive benchmark technique namely Random Search (RS). It is a baseline search technique that applied as a lowest benchmark to be compared against most of the search based metaheuristic algorithms [151].

- **RQ2. Different multi-objective optimization algorithms:** *How is our approach compared to the other two selected multi-objective evolutionary algorithms, MOCell and SPEA2?*

  Our approach is generic in which different multi-objective optimization algorithms can be used. Although NSGA-II is the main algorithm (described in details in Section 5.2.2 that we employed, there are also other suitable algorithms for our approach. We have tested our approach with two recently developed multi-objective evolutionary algorithms: Multiobjective Cellular Genetic Algorithm (MOCell) [161] and the Strength-based Evolutionary Algorithm (SPEA2) [483].
5.3. Evaluation

- **RQ3. Benefits from Multi-objective Approach:** Does our multi-objective approach provide more accurate and robust recommendation than alternative single-objective approaches?

To answer this question, we implemented the traditional single-objective genetic algorithms using reviewer expertise (RevExp) or the reviewer workload (RevWl) as the objective function. We name these alternative approaches as \textit{GA-RevExp} and \textit{GA-RevWl}. We then compare the performance of our proposed approach against these two single-objective approaches.

### 5.3.1 Datasets

In this section, we describe how data were collected for our empirical study and the experiments.

#### 5.3.1.1 Data collecting

To evaluate our approach, we have collected our data by selecting large software systems (projects) that utilize modern code review. Therefore, we choose to study the code review process of the four well-known open-source projects: LibreOffice, Android, Qt, and OpenStack systems. Android\footnote{https://source.android.com/} is one of the well-known open-source projects. It is a free software mobile operating system that is led and developed by Google. OpenStack\footnote{https://www.openstack.org/} is a cloud computing software platform that manages large pools of storage, networking resources, compute, and processing throughout a data center. Qt\footnote{https://www.qt.io/} is a framework of software cross-platform applications. LibreOffice\footnote{https://www.libreoffice.org/} is the most active community-driven open and free source software project developed by The Document Foundation.
5.3. Evaluation

Table 5.1: A statistical summary of the studied datasets for each studied project

<table>
<thead>
<tr>
<th>Project</th>
<th># Patches</th>
<th>#Reviewers</th>
<th># Patches/Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>36,771</td>
<td>2,049</td>
<td>1306/16198</td>
</tr>
<tr>
<td>LibreOffice</td>
<td>18,716</td>
<td>410</td>
<td>225/4972</td>
</tr>
<tr>
<td>Qt</td>
<td>65,815</td>
<td>1,238</td>
<td>1549/28457</td>
</tr>
<tr>
<td>OpenStack</td>
<td>108,788</td>
<td>3,734</td>
<td>12876/60661</td>
</tr>
<tr>
<td>Total</td>
<td>230,090</td>
<td>7,431</td>
<td></td>
</tr>
</tbody>
</table>

These projects have a large number of the patches being recorded in the code review tool (see Table 5.1). We use a review dataset that includes patches and developer information with their review discussion, the selected datasets have been used by other prior studies \([21, 23, 479, 481]\). We used the Representational State Transfer (REST) API provided by Gerrit\(^7\) to retrieve a list of reviewers and the information on the relevant reviewed patches.

We have collected a total of 230,090 patches from four projects and 7431 reviewers involved with those patches from October 2008 to November 2016.

5.3.1.2 Data pre-processing

We performed the pre-processing step to building the datasets for our experiments. Our approach utilizes two sets of information to make a recommendation: we selected the most recent 10% of the patches to build our ground-truth, and we use the remaining 90% of those patches for building a pool of candidate reviewers considering their attributes (i.e. metrics). After that, we cleaned the data by merging the duplicate accounts of candidate reviewers in order to assure the results accuracy. We identified the reviewer aliases email (i.e. aliases with multiple accounts with similar name or email) using an approach of \([488]\). To this end, we extracted five metrics from the datasets. Some of the extracted metrics are related to human factors (e.g. Number of Remaining Reviews, Familiarity between the Invited Reviewer and the Patch Author, \(\text{https://gerrit-review.googlesource.com/Documentation/rest-api.html}\).
5.3. Evaluation

and Review Participation Rate) while others are related to reviewer experience (e.g. Reviewer Code Authoring Experience and Reviewer Reviewing Experience). We then store the obtained data into a MySQL database.

In this study, we selected the relevant patches and we only collected the reviews that have been marked as “Merged” or “Abandoned”. For our data preparation, we excluded patches with the open status from the studied datasets. We then computed the metrics (i.e. attributes). To mimic this scenario, we collected all patches from each project (e.g. more than 100,000 patches were collected from the OpenStack project). Each of those patches has been assigned with a list of candidate reviewers.

In total, we conducted our study on 23,009 patches from four projects, which include 7431 reviewers. Table 5.1 summarizes the descriptive statistics for the patches and reviewers of our dataset in terms of a number of reviewers and patches per years, minimum, maximum, mean, median, and standard deviation. Across all the four projects, the number of patches per years varies. For example, the mean number of patches per year in OpenStack is 36263 patches, while it is only 6129 patches in Android.

5.3.2 Experimental settings and measures

This section describes the experimental setup and the performance evaluation used for the results analysis. In this work, each project has a number of patches (230,090 patches in total). We ran each algorithm 30 times for each patch, calculated the performance, and took the mean result.

5.3.2.1 Performance measures

To evaluate the performance of our proposed method, we employed precision, recall, and F-measure in Information Retrieval (IR) and commonly used for evaluating rec-
ommendation system [319,321,323,406]. Note that we measure the performance of
the model for each patch in a project, and then report the average performance across
all patches in the project. The performance measures can be defined as:

**Precision (Prec):** The ratio of the correctly recommended reviewers over all
the recommended reviewers. It is calculated as:

\[
Prec_i = \frac{|Actual_i \cap Recommended_i|}{|Recommended_i|}
\]

\[
Avg(Prec) = \frac{1}{m} \sum_{i=1}^{m} Prec_i
\]  \hspace{1cm} (5.5)

**Recall (Re):** The ratio of the correctly recommended reviewers over all the
actual reviewers. It is calculated as:

\[
Re_i = \frac{|Actual_i \cap Recommended_i|}{|Actual_i|}
\]

\[
Avg(Re) = \frac{1}{m} \sum_{i=1}^{m} Re_i
\]  \hspace{1cm} (5.6)

**F-measure (F1):** The weighted harmonic mean of the precision and recall. It
is calculated as:

\[
F1_i = \frac{2 \times (Prec_i \times Re_i)}{(Prec_i + Re_i)}
\]

\[
Avg(F1) = \frac{1}{m} \sum_{i=1}^{m} F1_i
\]  \hspace{1cm} (5.7)

Where \( m \) is the number of patches in a project, \( Recommended_i \) is a set of reviewers
recommended by an approach for a patch \( i \), and \( Actual_i \) is a set of reviewers actually
assigned for this patch \( i \).

We also use hypervolume [177] as a quality indicator for the volume of the space
covered by the non-dominated solutions. This measure has been used in previous work
(e.g. [14,324,325]) to as a performance indicator for multi-objective optimization. It
reflects the convergence and diversity of the solutions on a Pareto front (e.g. the higher
hypervolume, the better performance).

5.3.2.2 Statistical Test Methods

In order to compare the performance of two models, we employ the Wilcoxon Signed Rank Test \cite{435} to assess the statistical significance of the precision, recall, and F-measure achieved with the two models. The Wilcoxon Signed Rank Test does not assume a normal distribution in the data which is a safe test. The null hypothesis here is: "the performance provided by our approach is not different to those provided by alternative approaches", which we work to reject this null hypothesis. We set the confidence limit at 0.05 (i.e. $p < 0.05$). We then assessed whether the effect size is interesting by employing the correlated samples case of the Vargha and Delaney’s $\hat{A}_{XY}$ non-parametric effect size measure \cite{310}. The $\hat{A}_{XY}$ measures the probability that the performance achieved from model X is better than the performance achieved from model Y. Note that we have 3 performance measures: precision, recall, and F-measure. We thus employ the statistical testing and effect size testing on each individual measure (i.e. precision, recall, and F-measure) which can be defined as the following formula (let take recall as an example):

$$\hat{A}_{XY}(Re) = \frac{\#(X_{Re} > Y_{Re}) + (0.5 \times \#(X_{Re} = Y_{Re}))}{m}$$  (5.8)

Where $\#(X_{Re} > Y_{Re})$ is the number of patches that the recall (i.e. $Re_i$) from model X more than the recall from model Y, $\#(X_{Re} = Y_{Re})$ is the number of patches that the recall from model X equal to the recall from model Y, and $m$ is the number of patches. We then calculated $\hat{A}_{XY}(Prec)$ and $\hat{A}_{XY}(F1)$ from the same formula.
5.3. Evaluation

5.3.2.3 Parameter Setting

Our approach was implemented in the MOEA Framework\(^8\). We used the parameters that have been commonly used in previous search-based software engineering work \([14,326]\). Specifically, we employed tournament selection method and set the size of the initial population to 100. The number of generations was set to 100,000. Crossover probability was set to 0.9, mutation probability was 0.1, and reproduction probability was 0.2. We set the parameters \(\alpha_1, \alpha_2, \alpha_3, \alpha_4\) to 0.25 as default parameters.

5.3.3 Results

In order to assess the performance of our approach, we need to have the “true” ground-truths, i.e. a set of reviewers that are “best” to review a given patch at a specific time of a project. We were however unable to obtain those information since they were not available in the historical data that we extracted from the four projects we studied. Nonetheless, we were able to extract the reviewers who actually reviewed a given patch in our dataset. We found that on average the number of reviewers who actually reviewed a patch is very small (only around 2 reviewers per patch). Previous studies (e.g. \([414]\)) also confirm similar findings. The number of candidate reviewers were however very large, e.g. Android has 2,049 candidate reviewers while OpenStack has 3,734 reviewers. It is thus highly difficult to get the correct 2 reviewers out of 2,049. This is demonstrated from the results (Figure 5.6) of our approach using this ground-truths (we refer to as the original dataset). Our approach achieved only 0.2 (in Android), and 0.17 (in LibreOffice) in terms of precision, 0.35 (in Qt) and 0.28 (in Android) in terms of recall, 0.25 (in Qt) and 0.16 (in LibreOffice) in terms of F-measure. Similar results were also observed for the baselines and benchmarks we used for our dataset.

\(^8\)http://moeaframework.org/index.html
5.3. Evaluation

There is no guarantee that the reviewers who actually reviewed a patch were the best (optimal) ones at that given time in the project. We have thus explored a new set of ground-truths. The ground-truth of a given patch includes not only the reviewers who actually reviewed that patch but also the ones who reviewed other patches which changed the same files as this patch. The justification for this is that those reviewers were also suitable to review the patch in question, and thus should be included in the ground-truths. In the following, we will report our evaluation results\(^9\) to answer our research questions.

**Results for RQ1:**

Figures 5.7, 5.8, 5.9, and 5.10 report the boxplots to compare the performance achieved from our approach WLRRec against the random search method (RS) in terms of precision, recall, F-measure, and hypervolume. The analysis of all measures suggests that the recommendations obtained with our approach, WLRRec, are better than those achieved by using RS. WLRRec consistently outperforms RS in all 4 cases. Our approach improved between 174.72% (in ANDROID) to 319.75% (in OPENSTACK) in terms of precision, 67.28% (in OPENSTACK) to 303.02% (in ANDROID) in terms of recall, 189.09% (in LIBEROFFICE) to 222.18% (in QT) in terms of F-measure, and 87.56% (in QT) to 195.56% (in LIBEROFFICE) in terms of hypervolume over the random search method.

Table 5.2 shows the results of the Wilcoxon test and the corresponding \(\hat{A}_{XY}\) effect size to measure the statistical significance and effect size of the improved accuracy achieved by our approach over the random search. Our approach significantly outperforms the random search \(\left(p < 0.001\right)\) with effect sizes greater than 0.63 which can be considered as large effect size \(\left(\hat{A}_{XY} > 0.6\right)\), in all cases and for all three measures.

---

\(^9\)All the experiments were run on a Microsoft Windows 10 Home PC with an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz and 8.00 GB RAM.
Figure 5.6: Boxplots of results achieved by our WLRRec on the original dataset for the projects (Android, LibreOffice, Qt, OpenStack) in terms of precision, recall, f-measure, and hypervolume.

(a) Android

(b) LibreOffice

(c) Qt

(d) OpenStack
Our proposed approach, WLRRec, outperforms the random search in all four open source projects, thus passing the sanity check required by RQ1.

### Table 5.2: Comparison of WLRRec vs. RS, MOCell, and SPEA2 using Wilcoxon test and $\hat{A}_{xy}$ effect size (in brackets)

<table>
<thead>
<tr>
<th>Project</th>
<th>WLRRec vs.</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>Hypervolume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>MOCell</td>
<td>&lt;0.001 [0.55]</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.56]</td>
<td>&lt;0.001 [0.89]</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>&lt;0.001 [0.55]</td>
<td>&lt;0.001 [0.60]</td>
<td>&lt;0.001 [0.71]</td>
<td>&lt;0.001 [0.91]</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>&lt;0.001 [0.63]</td>
<td>&lt;0.001 [0.77]</td>
<td>&lt;0.001 [0.92]</td>
<td>&lt;0.001 [0.91]</td>
</tr>
<tr>
<td>LiberOffice</td>
<td>MOCell</td>
<td>&lt;0.001 [0.60]</td>
<td>&lt;0.001 [0.71]</td>
<td>&lt;0.001 [0.82]</td>
<td>&lt;0.001 [0.91]</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>&lt;0.001 [0.62]</td>
<td>&lt;0.001 [0.73]</td>
<td>&lt;0.001 [0.82]</td>
<td>&lt;0.001 [0.94]</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>&lt;0.001 [0.63]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.89]</td>
<td>&lt;0.001 [0.94]</td>
</tr>
<tr>
<td>Qt</td>
<td>MOCell</td>
<td>&lt;0.001 [0.80]</td>
<td>&lt;0.001 [0.64]</td>
<td>&lt;0.001 [0.73]</td>
<td>&lt;0.001 [0.92]</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>&lt;0.001 [0.80]</td>
<td>&lt;0.001 [0.65]</td>
<td>&lt;0.001 [0.74]</td>
<td>&lt;0.001 [0.93]</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>&lt;0.001 [0.85]</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.81]</td>
<td>&lt;0.001 [0.95]</td>
</tr>
<tr>
<td>OpenStack</td>
<td>MOCell</td>
<td>&lt;0.001 [0.74]</td>
<td>&lt;0.001 [0.74]</td>
<td>&lt;0.001 [0.71]</td>
<td>&lt;0.001 [0.81]</td>
</tr>
<tr>
<td></td>
<td>SPEA2</td>
<td>&lt;0.001 [0.72]</td>
<td>&lt;0.001 [0.75]</td>
<td>&lt;0.001 [0.68]</td>
<td>&lt;0.001 [0.80]</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>&lt;0.001 [0.79]</td>
<td>&lt;0.001 [0.77]</td>
<td>&lt;0.001 [0.79]</td>
<td>&lt;0.001 [0.89]</td>
</tr>
</tbody>
</table>

### Results for RQ2:
The boxplots of Figures 5.7, 5.8, 5.9, and 5.10 indicate the distribution of the results (precision, recall, F-measure, hypervolume) using different multi-objective optimization algorithms: MOCell, and SPEA2. In terms of MOCell, our approach using NSGA-II improved between 37.42% (in OPENSTACK) - 85.62% (in QT) precision, 49.35% (in Android) - 10538% (in OPENSTACK) recall, 46.22% (in ANDROID) - 98.41% (in QT) F-measure, and 21.44% (in QT) - 40.63% (in OPENSTACK) hypervolume.

NSGA-II improved over SPEA2 between 35.47% (in OPENSTACK) - 68.06% (in ANDROID) precision, 29.83% (in OPENSTACK) - 69.61% (in QT) recall, 41.76% (in OPENSTACK) - 102.64% (in QT) F-measure, and 29.34% (in QT) - 53.43% (in OPENSTACK) hypervolume.

The Wilcoxon test (see Table 5.2) also confirms that the improvement of our approach is significant ($p < 0.001$) with effect sizes greater than 0.55 in all cases.
5.3. Evaluation

Figure 5.7: Boxplots of results achieved by WLRRec, RS, MOCel1 and SPEA2 for Android project

(a) Precision

(b) Recall

(c) F-Measure

(d) Hypervolume
Figure 5.8: Boxplots of results achieved by WLRRec, RS, MOCCell and SPEA2 for LiberOffice project

(a) Precision

(b) Recall

(c) F-Measure

(d) Hypervolume
5.3. Evaluation

Figure 5.9: Boxplots of results achieved by WLRRec, RS, MOCell and SPEA2 for Qt project

(a) Precision

(b) Recall

(c) F-Measure

(d) Hypervolume
5.3. Evaluation

**Figure 5.10:** Boxplots of results achieved by WLRRec, RS, MOCell and SPEA2 for OpenStack project

(a) Precision

(b) Recall

(c) F-Measure

(d) Hypervolume
Figure 5.11: Evaluation results of WLRRec, the single objective approaches GA-RevEXP and GA-RevWL

(a) Android

(b) LiberOffice

(c) Qt

(d) OpenStack

Our proposed approach using NSGA-II significantly outperforms the two other alternative algorithms: MOCell, and SPEA2.

Results for RQ3:

Figure 5.11 also shows the results from using single-objective genetic algorithm: GA-RevEXP and GA-RevWL. WLRRec using multi-objective genetic algorithms improved between 63.28% (in OPENSTACK) - 104.26% (in ANDROID) in terms of
5.3. Evaluation

Table 5.3: Comparison of WLRREC vs. GA-RevWL and GA-RevWL using Wilcoxon test and $A_{XY}$ effect size (in brackets).

<table>
<thead>
<tr>
<th>Project</th>
<th>WLRRec vs.</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>GA-RevEXP</td>
<td>&lt;0.001 [0.78]</td>
<td>&lt;0.001 [0.81]</td>
<td>&lt;0.001 [0.81]</td>
</tr>
<tr>
<td></td>
<td>GA-RevWL</td>
<td>&lt;0.001 [0.78]</td>
<td>&lt;0.001 [0.81]</td>
<td>&lt;0.001 [0.81]</td>
</tr>
<tr>
<td>LiberoOffice</td>
<td>GA-RevEXP</td>
<td>&lt;0.001 [0.65]</td>
<td>&lt;0.001 [0.77]</td>
<td>&lt;0.001 [0.88]</td>
</tr>
<tr>
<td></td>
<td>GA-RevWL</td>
<td>&lt;0.001 [0.69]</td>
<td>&lt;0.001 [0.81]</td>
<td>&lt;0.001 [0.89]</td>
</tr>
<tr>
<td>Qt</td>
<td>GA-RevEXP</td>
<td>&lt;0.001 [0.90]</td>
<td>&lt;0.001 [0.72]</td>
<td>&lt;0.001 [0.89]</td>
</tr>
<tr>
<td></td>
<td>GA-RevWL</td>
<td>&lt;0.001 [0.90]</td>
<td>&lt;0.001 [0.79]</td>
<td>&lt;0.001 [0.80]</td>
</tr>
<tr>
<td>OpenStack</td>
<td>GA-RevEXP</td>
<td>&lt;0.001 [0.79]</td>
<td>&lt;0.001 [0.84]</td>
<td>&lt;0.001 [0.81]</td>
</tr>
<tr>
<td></td>
<td>GA-RevWL</td>
<td>&lt;0.001 [0.80]</td>
<td>&lt;0.001 [0.82]</td>
<td>&lt;0.001 [0.83]</td>
</tr>
</tbody>
</table>

precision, 72.25% (in OPENSTACK) - 169.45% (in ANDROID) in terms of recall, and 66.99% (in OPENSTACK) - 125.50% (in ANDROID) in terms of F-measure over the model using single-objective genetic algorithm: GA-RevEXP and GA-RevWL.

Across all projects, in terms of the precision measure, the highest mean value is 0.36 (in QT), and the highest median value is 0.32 (in QT). In terms of the recall, the OpenStack project is reported with the highest mean value of (0.52), and the highest median value of (0.5) in Qt. In terms of F-measure, the highest mean value is 0.42 (in LibreOffice), and the highest median value is 0.38 (in Android). The hypervolume indicates the highest mean value (0.8) in Qt and the highest median value (0.9) in Android.

The Wilcoxon test also show the comparison between our Multi-objective WLR-Rec and single-objective (GA-RevEXP & GA-RevWL) approaches. In Table 5.11 it can be seen that, the improvement of the multi-objective approach MOSBIP over the single-objective is significant ($p < 0.001$) in all cases with the effect size greater than 0.55 all cases.

Using multi-objective approach provides more accurate and robust recommendation than single-objective models.
Our approach performs better than the other approaches because of its elite strategy which is based on evolving the candidate solutions to gain and maintain a well-distributed set of near-optimal solutions, namely the Pareto front (non-dominated solutions) for solving a multi-objective optimization problem. The elitism of the non-dominated solution provides a proper trade-off between all optimized objectives considering all objectives (i.e. without ignoring any objective). To this end, an extensive evaluation on four large projects including Android, LibreOffice, Qt, and OpenStack, demonstrates that our approach is capable of recommending reviewers who have high reviewing experience with a low reviewing workload. Our recommendations aim to create a balancing workload across the developers while ensuring them be assigned to relevant patches. Our contribution advances the state-of-art of in code reviewer recommendations, helping improve the effectiveness and efficiency of the code review process.

5.3.4 Threats to validity

To mitigate threats to construct validity, we have collected our data by selecting large software systems (projects) that utilize modern code review. We have used a review dataset that includes patches and developer information with their review discussion to build our ground truth. We carefully followed recent best practices to minimize conclusion instability [208, 227, 328]. We selected unbiased error measures and applied a number of statistical tests to verify our assumptions [329]. Our study was performed on four datasets of different sizes. Furthermore, we applied the widely used metrics (precision, recall, and F-measure) to evaluate the effectiveness of our proposed approach. These metrics have been used in several software engineering studies [330, 331]. We also applied the hypervolume measure since it has been successfully used in related research work [332]. To overcome the external validity threat, we have considered 23,009
patches from four projects, which include 7,431 reviewers. The number of patches and the related candidate reviewers in those four large software projects candidates is significantly diverse. All patches and associated reviewers reports are real data. These data are generated from open source settings in software development. However, we cannot claim that our datasets are representative of all kind of software projects and that our results can generalize to all software projects especially those in commercial settings.

5.4 Related work

Solving the problem of reviewers recommendation by selecting appropriate reviewers is highly important to improve the review quality. For example, [21] raise concerns that up to 30% of patches of the Qt project have the difficulty in finding an appropriate reviewers, which often delay the code review and integration process.

There are various factors that have been used when recommending an appropriate reviewer. For example, [220] leveraged a change history of a patch that is reviewed by reviewers in the past to recommend reviewers. [21] proposed a RevFinder approach by leveraging the file review history to recommend a suitable reviewer. The reviewer recommendation process uses the modified information (i.e, reviewers comments) of a patch in the file path of the project, they computed reviewers expertise from the similarity (in terms of the modified information) between a new patch and prior patch that had been already reviewed by reviewers. Ultimately, this process will support developers to identify the highly recommended reviewer. The work in [24] proposed chRev approach based on reviewer expertise. This approach considered the historical contributions of reviewers (i.e., the review count of developers to source files) to compute the expertise of the reviewers and thereby to recommend suitable reviewers. The expertise is computed from the previously made comments on the code.
5.4. Related work

being reviewed, the time that has been spent on those comments (i.e., workdays), and
the time since the last comment.

A study by [468, 469] addressed the challenge of reviewer selection in reviewer
recommender of pull–requests in Github. The authors assigned each developer with
expertise score. They have computed reviewer expertise and common interests between
reviewers and patch authors. They measure the cosine similarity by using textual
semantic and comment network amongst developers (patch authors and reviewers).
Ultimately, all pull requests (new) are scored with expertise score and then assigned
to reviewers. In addition, the work in [219] argued other important factors such as time,
interests, and priorities can have an impact on the reviewer recommendation process.
The study reported in [23] also suggested that human factors play an essential role
in the decision of reviewer participation. Other studies [20, 222, 472] reported other
factors like the number of reviewers and the size of the patches might also have an
impact on the amount of review participation. They found that the lack of review
participation can negatively impact on the quality of the code review process in the
long term.

The work in [473] investigated the impact of the non-technical factors and human
aspects on the code review outcome. They described that factors like patch author
experience and bug priority could affect the code review process in a significant way.
Recently, [221] have described the relationship between patch characteristics and re-
view participation. The authors discussed that technical factors such as patch size
might lead to a lack of review participation (i.e., poor review). They focused on patch
characteristics that receive slow initial feedback and do not attract reviewers.

Recently, the work in [22] raised concerns that manually selecting reviewers for
a code change is still a costly and time-consuming task. The authors proposed a
RevRec approach to support the decision making for the path submitters and to find
the most appropriate reviewers to be assigned for a code change. They used the genetic
algorithm (GA) to consider both reviewer expertise and reviewer collaboration in their
proposed search-based optimization problem.

Selecting the right reviewers is one of the main challenges that patch authors can
face in the code review process, particularly, when the process is manually performed.
However, there is no prior studies consider the reviewer workload when recommending
reviewers. Thus, in this chapter, we formulate the reviewer recommendation as a
multi-objective evolutionary approach to recommend reviewers, while balancing the
workload among reviewers.

The contributions of our chapter are as follows:

1. We introduce a multi-objective search-based evolutionary approach, namely,
   WLRRec to find the most appropriate reviewers. Our approach considers maxi-
mizing the reviewers expertise to achieve successful code review while balancing
   the workload among reviewers (i.e. minimize the workload difference between
developers).

2. We evaluate our approach using other multi-objective evolutionary search al-
gorithms: Multiobjective Cellular Genetic Algorithm (MOCell) \[161\] and the
   Strength-based Evolutionary Algorithm (SPEA2) \[317\] on four large open source
   projects (Android, LiberOffice, Qt, and OpenStack). Our results indicate our
   approach significantly outperforms the MOCell and SPEA2.

3. As a second empirical study, we compare the performance of our proposed ap-
   proach (WLRRec) against the two single-objective approaches, genetic algo-
   rithms using reviewer expertise (RevExp) or the reviewer workload (RevWl).
   The results indicate that WLRRec significantly outperforms the single-objective
   approaches.
5.5 Chapter summary

We have proposed a multi-objective search-based evolutionary approach (WLRRec) to support the developers (patch authors) in selecting the most appropriate reviewers for a submitted patch (i.e., code changes). The search is guided simultaneously by two objectives: maximizing the reviewer’s expertise to achieve successful code review while balancing the workload among reviewers (i.e., minimize the workload difference between developers).

An extensive evaluation performed on four large open source projects (Android, LibreOffice, Qt, and OpenStack). The empirical evaluation results show that our approach can identify reviewers who have high reviewing experience with a low reviewing workload. Our results indicate that our approach significantly outperforms random search in all projects. The evaluation also demonstrates the advantages of using a multi-objective approach over single-objective approaches. We also tested our approach with several multi-objective evolutionary algorithms and found that WLRRec was generally the best performers in this context. Our contribution advances the state-of-art of reviewer recommendations which in turn will increase the effectiveness and efficiency of code review processes. Future work would involve validating these results with additional projects, especially those in commercial settings. We will also explore the use of other multi-objective evolutionary algorithms as part of our future work.
Chapter 6

Conclusions and future work

"The mind that opens to a new idea never returns to its original size."

Albert Einstein

This chapter summarises a number of contributions that have been made throughout the development of this thesis and provide some directions for future research.

6.1 Summary of contributions

The main objective of this thesis is to provide automated support which can help decision-makers at different stages of the software development process. To this end, we developed novel approaches using multi-objective search-based evolutionary to select candidate issues for the upcoming iteration, estimate issue resolution time and effort, and to recommend suitable reviewers for code changes. We adapted a meta-heuristic technique, namely genetic algorithms, to generate a large number of candidate selection of issues, estimation models, and reviewers and search for the ones that are optimal with respect to a number of objectives. The contributions of this thesis are summarized
6.1. Summary of contributions

as follows:

- **Iteration planning in agile development (Chapter 3):**
  
  We employed a multiobjective search-based evolutionary approach to develop machinery which supports agile teams in selecting issues to be completed in an upcoming iteration. We explored two objectives which guide our search algorithms. The first objective is to maximize the business value that an iteration delivers to the customers. Each iteration has a goal which is defined by the team to state what will be accomplished during an iteration. The iteration goal is usually a short text description. Our second objective is to select issues that maximize this collective contribution towards the iteration’s goal. An extensive evaluation was performed against a dataset of 233 iterations (which consist of 55,662 issues in total) we collected from six different Apache projects. The results from our experiments show that our approach has consistently outperformed random guessing in all projects. The evaluation also demonstrates the advantages of using a multi-objective approach over single-objective approaches. In addition, our approach was also tested with several multi-objective evolutionary algorithms to identify best performers in this context.

- **Issue effort and time estimation (Chapter 4):**
  
  We applied our proposed approach to predict the effort (in terms of both resolution time and story points) of each single issue in a software project. Our search is simultaneously guided by two objectives. The first objective is to minimize the Sum of Absolute Errors, which measures the accuracy of an estimation model in terms of the differences between values (i.e. issue resolution time and/or story points) estimated by the model and the values actually observed. The pressure of minimizing the estimation errors may, however, cause the solution model to adhere precisely to noisy data in the training set, which potentially make the
model be excessively large and complex (hence, overfitting problems). Our sec-
ond objective is to minimize the complexity of an estimation model, which can be
measured in terms of the size of an expression tree representing the model. This
second objective also leads to reduced computational costs since it encourages
parsimonious (thus, computationally efficient) candidate solutions be generated.

In terms of the resolution time estimation, we collected 8,260 issues from five
different projects. In terms of the story points estimation, from eight differ-
ent projects, we collected 4,677 issues. The evaluation results on 12,937 issues
from 13 large open source projects demonstrate that our approach has a strong
predictive performance in effort prediction.

• Workload-Aware Code Reviewer Recommendation (Chapter 5):

We have developed a workload-aware reviewer recommendation approach. To
this end, we employed our approach to find appropriate reviewers while consider-
ing their reviewing workload. Our work investigated two objectives which guide
our search algorithms. The first objective is to maximize the reviewers expertise
to ensure the most relevant reviewers are assigned to relevant a patch. During
the code review process, some of the assigned reviewers cannot contribute due
to neglecting their current workload. Hence, our second objective is to minimize
the workload difference between developers. The evaluation results indicate that
our search-based approach consistently outperforms the common baselines. We
also demonstrate the effectiveness of using a multi-objective approach against
the single objective approach. The evaluation was performed against a dataset
of 230,090 patches (which consist of 7431 reviewers in total) we collected from
four different open-source projects (Android, LiberOffice, Qt, and OpenStack).
6.2 Future Work

In this section, we discuss the potential future directions related to the research work carried out in this thesis.

- **Industry acceptance**: The proposed approach can be developed into a tool in order to be integrated into industrial case studies. Several studies (e.g. 459, 461) in industrial experiences have discussed the need for reliable and usable tools. Developing our models and technique into a prototype tool is part of our future work. Future work also involves the validation of these tools by industrial practitioners.

- **Commercial settings**: For our empirical studies, we used real-world data large open source projects. However, we cannot claim that our datasets are representative of all kind of software projects and that our results can generalize to all software projects. Expanding the study to other large open source software projects, especially to commercial settings would be part of future work.

- **Using different data sources**: Our proposed approaches leverage the data collected from the issue tracking system (Jira) and other software repositories (e.g. Android, LibreOffice, OpenStack, and Qt). The extent of future work aims to consider other aspects (e.g. skilled human resource allocation [458]) using other various datasets in software repositories, aiming to build different predictive models. For example, diverse real-life effort data from the version control system (e.g. GitHub) can be used to enhance effort estimation models.

- **Explore more advanced algorithms**: The continuous development of different emerging software engineering areas put the software engineers and other decision-makers in a challenge with a series of complex decision scenarios. Therefore, future work will consider scaling up our approach in order to deal with such
6.2. Future Work

scenarios. This can lead to the new opportunity of exploring different search models and optimization techniques. For example, future work would involve investigating the use of other objectives such as confidence interval and other complexity measures (e.g. order of nonlinearity). Exploring the use of other multi-objective evolutionary algorithms (e.g. NSGA-III) as part of our future work.
References


[138] Robert M Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip: Optimal product selection from feature models using many-objective evolution-


[183] Rafael Munoz-Salinas, Eugenio Aguirre, Oscar Cordon, and Miguel Garcia-Silvente. Automatic tuning of a fuzzy visual system using evolutionary algo-


[198] Yasutaka Kamei, Shinsuke Matsumoto, Akito Monden, Ken-ichi Matsumoto, Bram Adams, and Ahmed E Hassan. Revisiting common bug prediction find-


References


[286] Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, Mel Ó Cinnéide, and Kalyanmoy Deb. On the use of many quality attributes for software refac-


[324] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M German, and Katsuro Inoue. Search-based software library recommendation


References


[440]


[448] Yuanyuan Zhang, Mark Harman, and A Mansouri. The sbse repository: A repository and analysis of authors and research articles on search based software engineering. crestdweb. cs. ucl.ac.uk/resources/sbse repository, 2012.


[472] Peter C Rigby, Daniel M German, Laura Cowen, and Margaret-Anne Storey. Peer review on open-source software projects: Parameters, statistical mod-


[479] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, AE Cruz, Kenji Fujiwara, and Hajimu Iida. Who does what during a code review? datasets of


