

Management & Recommendation of Code Review using Multi-Objective Optimization Algorithm



Author:

Zaeem Anwaar

MS19-00000320966

Supervisor:

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

December 2022

Management & Recommendation of Code Review using Multi-Objective Optimization Algorithm

Author:

Zaeem Anwaar

00000320966

A thesis submitted in partial fulfillment of the requirements for the degree of
MS Software Engineering

Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature: _____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

December 2022

Declaration

I certify that this research work titled “*Management & Recommendation of Code Review using Multi-Objective Optimization Algorithm*” is my work. The work has not been presented elsewhere for assessment. The material that has been used from other sources; has been properly cited/acknowledged/referred to.

Signature of Student

Zaeem Anwaar

00000320966

PLAGIARISM REPORT (TURNITIN REPORT)

This thesis copy has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Signature of Student

Zaeem Anwaar

00000320966

Signature of Supervisor

Dr. Wasi Haider Butt

Language Correctness Certificate

This thesis has been read by an English expert and is free of most typing, syntax, semantic, grammatical, and spelling mistakes. The thesis is set all according to the format given by the university.

Signature of Student

Zaem Anwaar

00000320966

Signature of Supervisor

Dr. Wasi Haider Butt

Copyright Statement

- Copyright in the text of this thesis rests with the student author. Copies (by any process) either in full or of extracts, may be made only per instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

Acknowledgments

I am extremely thankful to ALLAH Almighty for his bountiful blessings throughout this work. It was quite an arduous exercise that could not have been completed without the help of God Almighty and the strength that he bestowed upon me.

I would like to offer my sincere thanks to my incredible supervisor '**Dr. Wasi Haider Butt**' and the entire committee: '**Dr. Arslan Shaukat**' and '**Dr. Ahsan Shahzad**'. Without their support – in every sense of the word – and constant guidance, this thesis would never have been conceived or completed. I cannot thank them enough for their role in the completion of this thesis and report.

I am also eternally grateful to fellows and colleagues, especially my batchmates (M. Usman Farooq, Mehwish Naz, and Farasat Ullah Khan), who helped me and supported me to accomplish this herculean task.

*Dedicated to my mother, whose tremendous continuous support and
endless prayers led me to this accomplishment*

Abstract

Background: Software code review is a one of the major and important activity in modern software development and evolution. To improve software quality, identify and remove defects before integration, code review is considered as efficient and effective practice. Code reviewers having right expertise, experience and apt amount of knowledge with the code being reviewed leads to successful code processes, fewer bugs and less maintenance cost.

Aim & Objective: Usually existing studies identify code reviewers based on one or two objectives i.e., expertise, availability etc. to review pull requests. With the growing size of distributed development teams, picking suitable reviewers is a challenging task. However, due to less resources and shorter deadlines, the management of code reviews and appropriate recommendation of code reviewers based on three objectives consecutively is an ambitious task to be considered as aim of this thesis.

Methodology: This thesis addresses the formulation for managing and recommending code reviewers based on multi conflicting objectives (i.e., availability, expertise and collaboration) simultaneously. ‘NSGA-III’ is used as optimization algorithm to find the most suitable reviewers while keeping expertise and availability ratio high and less collaboration between reviewers and developers.

Results and Conclusion: The results were implemented and validated on three (medium to large size) open-source projects named as LibreOffice, Qt and OpenStack. We calculated precision, recall, MRR, accuracy for all 3 projects on average. The results from our proposed approach accurately recommended the code reviewers with the precision up to 80%, 86% of recall, 82% mean reciprocal rank and 84% average accuracy by improving state-of-the-art. We also compared the experimental sets between NSGA-III and NSGA-II in terms of finding mean fitness and execution time of both algorithms. As a result, NSGA-III recommended the reviewers in less execution time and better fitness values in comparison to NSGA-II in all experimental sets. The proposed approach could be practical to MCR in order to help developers while recommending suitable code-reviewers in less time and resources to speed up the review process.

Keywords: Code-Reviewer Recommendation, Modern Code Review, Modern Software Development, Multi-Objective Algorithm, NSGA-III

Table of Contents

DECLARATION.....	I
PLAGIARISM REPORT (TURNITIN REPORT).....	II
LANGUAGE CORRECTNESS CERTIFICATE.....	III
COPYRIGHT STATEMENT.....	IV
ACKNOWLEDGMENTS.....	V
ABSTRACT.....	VII
LIST OF FIGURES.....	X
LIST OF TABLES.....	XII
CHAPTER 1.....	1
CHAPTER 1 : INTRODUCTION.....	2
1.1 MOTIVATION.....	3
1.2 PROBLEM STATEMENT.....	4
1.3 AIMS AND OBJECTIVES.....	4
1.4 THESIS OUTLINE.....	5
CHAPTER 2.....	6
CHAPTER 2 : CODE REVIEWER RECOMMENDATION.....	7
2.1 REVIEW PROCESS & CRR.....	7
2.2 MULTI-OBJECTIVE OPTIMIZATION.....	8
CHAPTER 3.....	10
CHAPTER 3 : LITERATURE REVIEW.....	11
3.1 OVERVIEW AND MAJOR OUTCOMES OF SLR.....	11
3.2 REVIEW METHODOLOGY.....	13
3.2.1 <i>Research Questions</i>	13
3.2.2 <i>Category Definition</i>	13
3.2.3 <i>Review Protocol</i>	15
3.3 RESULTS, ANALYSIS & ANSWERS TO RESEARCH QUESTIONS.....	21
3.4 CONCLUSION TO LITERATURE REVIEW.....	29
3.5 SUMMARY TABLE OF LITERATURE REVIEW.....	30
3.6 RESEARCH GAP.....	32
CHAPTER 4.....	34
CHAPTER 4 : PROPOSED APPROACH.....	35
4.1 DATA PRE-PROCESSING.....	36
4.1.1 <i>Conversion of SQL & JSON to .CSV Format</i>	36
4.1.2 <i>Removal of NaN Values</i>	37
4.1.3 <i>Timestamp Data</i>	37
4.1.4 <i>Normalization</i>	38
4.2 CODE REVIEWER RECOMMENDATION (CRR) FRAMEWORK.....	39
4.2.1 <i>Major Components of the Approach</i>	39
4.2.1.1 <i>Reviewer Expertise Matrix</i>	39
4.2.1.2 <i>Reviewer Developer Collaboration Matrix</i>	40
4.2.1.3 <i>Reviewer Availability</i>	40
4.2.2 <i>Multi-Objective Optimization (NSGA-III)</i>	40
4.2.3 <i>Fitness Functions</i>	42
4.2.3.1 <i>Availability</i>	43
4.2.3.2 <i>Expertise</i>	43

4.2.3.3	Collaboration	44
CHAPTER 5	45
CHAPTER 5 : IMPLEMENTATION, RESULTS & DISCUSSION	46
5.1	DATA COLLECTION/DATASET	46
5.1.1	SQL Format.....	47
5.1.2	CSV Format.....	47
5.1.3	Postman API Platform	47
5.2	EXPERIMENTAL SETUP AND RESEARCH QUESTIONS.....	48
5.2.1	Research Questions.....	49
5.2.2	Answers to Research Questions	54
5.3	COMPARISON OF FRAMEWORK USING LIBREOFFICE DATASET	56
5.3.1	Comparison of visualization of 'Availability' (NSGA-II vs NSGA-III).....	56
5.3.2	Comparison of visualization of 'Collaboration' (NSGA-II vs NSGA-III).....	59
5.3.3	Comparison of visualization of 'Expertise' (NSGA-II vs NSGA-III).....	61
5.3.4	Comparison of execution time (NSGA-II vs NSGA-III).....	63
5.4	COMPARISON OF FRAMEWORK USING QT PROJECT	64
5.4.1	Comparison of visualization of 'Availability' (NSGA-II vs NSGA-III).....	64
5.4.2	Comparison of visualization of 'Collaboration' (NSGA-II vs NSGA-III).....	66
5.4.3	Comparison of visualization of 'Expertise' (NSGA-II vs NSGA-III).....	69
5.4.4	Comparison of execution time (NSGA-II vs NSGA-III).....	71
5.5	COMPARISON OF FRAMEWORK USING OPENSTACK PROJECT	71
5.5.1	Comparison of visualization of 'Availability' (NSGA-II vs NSGA-III).....	71
5.5.2	Comparison of visualization of 'Collaboration' (NSGA-II vs NSGA-III).....	73
5.5.3	Comparison of visualization of 'Expertise' (NSGA-II vs NSGA-III).....	76
5.5.4	Comparison of execution time (NSGA-II vs NSGA-III).....	78
5.5.5	Abnormal / Premature Behavior of Algorithm (NSGA-II vs NSGA-III) for Experiment no 7 (OpenStack Project).....	78
5.6	LIMITATIONS.....	79
CHAPTER 6	80
CHAPTER 6 : CONCLUSION & FUTURE WORK	81
6.1	CONCLUSION.....	81
6.2	FUTURE WORK	81

List of Figures

Figure 1. Thesis Outline.....	5
Figure 2. The MCR Process [5]	7
Figure 3. Example of code review (LibreOffice).....	8
Figure 4. Genetic Algorithm Flowchart.....	9
Figure 5. Overview & Major Outcomes of SLR.....	11
Figure 6. Search Process	17
Figure 7. Distribution of Selected Studies based on Publication Year	19
Figure 8. Research Productivity of Algorithms (2011~2022)	28
Figure 9. Overview of multi-objective search-based approach for CRR.....	35
Figure 10. SQL to .CSV Format	36
Figure 11. JSON to .CSV Format	36
Figure 12. NaN Value (left) and Removed NaN Values (right)	37
Figure 13. Date-Time Data (left) and Timestamp formatted data (right)	38
Figure 14. Un-normalized data (left) and Normalized data (right).....	39
Figure 15. An example of Postman Data Collection	47
Figure 16. Comparison of MRR for Projects.....	54
Figure 17. Average Accuracy of Projects	54
Figure 18. LibreOffice Availability Best Solution Experiment-1	57
Figure 19. LibreOffice Availability Best Solution Experiment-2	57
Figure 20. LibreOffice Availability Best Solution Experiment-3	57
Figure 21. LibreOffice Availability Best Solution Experiment-4	58
Figure 22. LibreOffice Availability Best Solution Experiment-5	58
Figure 23. LibreOffice Availability Best Solution Experiment-6	59
Figure 24. LibreOffice Collaboration Best Solution Experiment-1.....	59
Figure 25. LibreOffice Collaboration Best Solution Experiment-2.....	59
Figure 26. LibreOffice Collaboration Best Solution Experiment-3.....	60
Figure 27. LibreOffice Collaboration Best Solution Experiment-4.....	60
Figure 28. LibreOffice Collaboration Best Solution Experiment-5.....	60
Figure 29. LibreOffice Collaboration Best Solution Experiment-6.....	61
Figure 30. LibreOffice Expertise Best Solution Experiment-1	61
Figure 31. LibreOffice Expertise Best Solution Experiment-2	62
Figure 32. LibreOffice Expertise Best Solution Experiment-3	62
Figure 33. LibreOffice Expertise Best Solution Experiment-4	62
Figure 34. LibreOffice Expertise Best Solution Experiment-5	63
Figure 35. LibreOffice Expertise Best Solution Experiment-6	63
Figure 36. Qt Availability Best Solution Experiment-1	64
Figure 37. Qt Availability Best Solution Experiment-2	65
Figure 38. Qt Availability Best Solution Experiment-3	65
Figure 39. Qt Availability Best Solution Experiment-4	65
Figure 40. Qt Availability Best Solution Experiment-5	66
Figure 41. Qt Availability Best Solution Experiment-6	66
Figure 42. Qt Collaboration Best Solution Experiment-1	67
Figure 43. Qt Collaboration Best Solution Experiment-2	67
Figure 44. Qt Collaboration Best Solution Experiment-3	67
Figure 45. Qt Collaboration Best Solution Experiment-4	68
Figure 46. Qt Collaboration Best Solution Experiment-5	68
Figure 47. Qt Collaboration Best Solution Experiment-6	68

Figure 48. Qt Expertise Best Solution Experiment-1	69
Figure 49. Qt Expertise Best Solution Experiment-2	69
Figure 50. Qt Expertise Best Solution Experiment-3	69
Figure 51. Qt Expertise Best Solution Experiment-4	70
Figure 52. Qt Expertise Best Solution Experiment-5	70
Figure 53. Qt Expertise Best Solution Experiment-6	70
Figure 54. OpenStack Availability Best Solution Experiment-1	71
Figure 55. OpenStack Availability Best Solution Experiment-2.....	72
Figure 56. OpenStack Availability Best Solution Experiment-3.....	72
Figure 57. OpenStack Availability Best Solution Experiment-4.....	72
Figure 58. OpenStack Availability Best Solution Experiment-5.....	73
Figure 59. OpenStack Availability Best Solution Experiment-6.....	73
Figure 60. OpenStack Collaboration Best Solution Experiment-1	74
Figure 61. OpenStack Collaboration Best Solution Experiment-2.....	74
Figure 62. OpenStack Collaboration Best Solution Experiment-3.....	74
Figure 63. OpenStack Collaboration Best Solution Experiment-4.....	75
Figure 64. OpenStack Collaboration Best Solution Experiment-5.....	75
Figure 65. OpenStack Collaboration Best Solution Experiment-6.....	75
Figure 66. OpenStack Expertise Best Solution Experiment-1	76
Figure 67. OpenStack Expertise Best Solution Experiment-2.....	76
Figure 68. OpenStack Expertise Best Solution Experiment-3.....	76
Figure 69. OpenStack Expertise Best Solution Experiment-4.....	77
Figure 70. OpenStack Expertise Best Solution Experiment-5.....	77
Figure 71. OpenStack Expertise Best Solution Experiment-6.....	77
Figure 72. LibreOffice Abnormal / Premature Behavior of Algorithm NSGA-II.....	78
Figure 73. LibreOffice Abnormal / Premature Behavior of Algorithm NSGA-III.....	78

List of Tables

Table 1. Categorization of Review Information	13
Table 2. Search Keywords and Results.....	17
Table 3. Accepted Research Papers Vis a Vis Databases	19
Table 4. Year-wise distribution of Selected Studies	20
Table 5. Template of Data Extraction and Synthesis.....	21
Table 6. Leading Code Reviewer Recommendation Tools	22
Table 7. Leading CRR Validation Protocols	23
Table 8. Optimization Algorithms	24
Table 9. Threats to Validity for CRR Tools	26
Table 10. Ranking of CRR Tools based on Accuracy	29
Table 11. Summary Table of Literature Review	30
Table 12. Data Normalization.....	39
Table 13. Summary Table of Open-Source Validation Projects.....	48
Table 14. Experimental Genetic Operators.....	49
Table 15. Precision Results.....	52
Table 16. Recall@Exp Results	53
Table 17. Mean Fitness of Availability (LibreOffice)	59
Table 18. Mean Fitness of Collaboration (LibreOffice)	61
Table 19. Mean Fitness of Expertise (LibreOffice)	63
Table 20. Comparison of Execution time for LibreOffice Project	63
Table 21. Mean Fitness of Availability (Qt).....	66
Table 22. Mean Fitness of Collaboration (Qt)	68
Table 23. Mean Fitness of Expertise (Qt).....	70
Table 24. Comparison of Execution time for Qt Project	71
Table 25. Mean Fitness of Availability (OpenStack)	73
Table 26. Mean Fitness of Collaboration (OpenStack)	75
Table 27. Mean Fitness of Expertise (OpenStack)	77
Table 28. Comparison of Execution time for OpenStack Project.....	78

Chapter 1

INTRODUCTION

Chapter 1 : INTRODUCTION

Software code review is an integral part of software development and has been in practice for more than three decades [1]. It involves identifying and fixing the defects i.e., logical errors or bugs in a software system, to ensure code quality. The code review process begins when a changed source code is submitted for code inspection. Numerous studies claim that the quality of software systems can be enhanced by reducing defects during code inspection [2]. A reviewer is requested to review the change and identify the issues with the change, and then recommend further actions to the software developer responsible for the code. A series of meetings between the reviewer and the developer are taken place, to ensure a mutual understanding of the change made and the review feedback. The drawback of this manual process of code review is time-consuming and expensive; as the time, effort and experience of the reviewers are not judiciously and efficiently utilized. For example, if a request is routed to a reviewer who is already committed or who doesn't possess the requisite expertise required to meet this specific request, the outcome would be a waste of time and effort and a 'quality review' may not be expected [3].

Code review is a challenging process because it requires dedicated efforts of a reviewer to read, comprehend and provide actionable feedback on a code change. The purpose is to get a code check-in the shortest possible time as the author/company needs to release the new version of the modified code as early as possible. Thus, a suitable code reviewer is required to serve the purpose of code review. Such a reviewer should possess thorough knowledge, experience and expertise needed for the job and shouldn't be overcommitted as well. Only, the reviewer with the requisite expertise and appropriate time may contribute toward efficient examination of the code changes and defects [2], [3]. This process of selection of appropriate reviewers may be simple for smaller software firms dealing with or working on a fewer number of projects. However, professionally competent and larger firms with multiple ongoing projects face greater difficulties in selecting appropriate reviewers due to the inherent issues [4].

Over the period, the manual process of assigning a reviewer has evolved into an informal, fully automated, structured and documented approach [5]. It has progressed to a lightweight, quicker and tool-based process named Modern Code Review (MCR) [3]. MCR is also known as change-based code review [4]. MCR is a collaborative, quicker and automated approach that ensures that both author and code reviewer follow the standards of code review in a literal manner. Here, the reviewer is assigned to review a specific code, based on certain logic

and certain credentials, in an automated manner. Some of the benefits of choosing the reviewer in an automated manner are: 1) Reviewer is automatically assigned as per certain credentials in a much shorter time frame without compromising other projects. 2) Human factor / biases which may result in a selection of inappropriate reviewers is eliminated. 3) All the reviewers share the optimized load. 4) The Selection of the right reviewer (as-per requisite skills, experience and commitment) enhances the quality of the review [6], [7].

Most of the previously proposed tools for Code Reviewer Recommendation (CRR) use the credential “reviewer expertise” to recommend [8]. Reviewer expertise is the gained knowledge of the changes that had been under review or are currently under review. It could be explained as the reviewer having high expertise should have reviewed the similar files/code changes in the past or lately [4]. But the availability or delay in work (workload) of reviewers having high expertise can’t be made sure every time while assigning them.

This **thesis proposes** to articulate the selection of peer code reviewers as a multi-objective problem to address the challenges discussed above. The multi-objective problem is defined as “to balance between more than one conflicting objective/multiple objectives i.e., expertise, availability & collaboration” [9]. After performing detailed literature, we adopted one of the infrequently or less used multi + many objective search algorithms, NSGA-III, to find the balance in context to our 3 objectives. The approach navigates between three different proportions by providing multiple non-dominant peer reviewer recommendations instead of one solution as explained mostly in literature. Moreover, we validated our approach on 3 open-source projects to confirm its efficiency and execution time to comparison to the state of the art.

This **introduction chapter summarizes** that software code review is one of the most significant software activities executed for code inception to lessen defects/bugs and to improve the quality of software by automated recommendation of the right reviewer for the right code file. Automated tools take certain credentials or levels i.e., expertise, availability and collaboration to recommend reviewers. We used NSGA-III (multi + many) [10] objective search algorithm to provide multiple non-dominant peer reviewers.

1.1 Motivation

The research motivation is to recommend/choose the right reviewers for the code review process more quickly and accurately to save time and resources. The research is aligned with the automated selection of reviewers based on certain credentials for the betterment of

software systems. Choosing the appropriate reviewer in an automated manner in terms of a multi-objective problem will recommend more than one solution in a much shorter time frame without compromising other projects. Human biases that may result in a selection of inappropriate reviewers will be eliminated. Also, all the reviewers would share the optimized load. Lastly, the selection of the right reviewer (as-per requisite skills, experience and collaboration) would enhance the quality of the review.

1.2 Problem Statement

Automated code reviewer recommendation contributes a vital role in the Modern code review. Researchers have proposed different approaches/tools that recommend the reviewers based on some credentials i.e., level of expertise. But these tools recommend multiple reviewers based on a single credential usually. To deal with a large number of possible reviewers for multiple pull requests in terms of multi-objective context is a management problem that is under-studied in the research literature. This management process requires handling multiple competing criteria including expertise, availability and previous collaborations with the owners and reviewers. To overcome these issues, this thesis presents an approach to articulate the selection of peer code reviewers as a multi-objective problem. The approach navigates between three different proportions by providing multiple non-dominant peer reviewer recommendations instead of one solution. Additionally, the purpose is not only to recommend reviewers, the idea is to manage code reviews with shorter deadlines and limited resources while keeping expertise and availability highlighted.

1.3 Aims and Objectives

The major objectives of the research are as follows:

- To reform and organize 3 different open-source project datasets, that are to be used for CRR.
- To perform a detailed systematic literature review of the recent research on CRR
- Explore Multi-objective optimization search algorithms.
- Propose an approach that navigates between three different proportions/credentials by providing multiple non-dominant peer reviewer recommendations instead of one solution.
- Analyzing and validating the precision, recall, MRR and accuracy of the proposed

approach.

- Comparison of the results with state-of-the-art.

1.4 Thesis Outline

This remaining work is structured as follows:

Chapter 2 covers the basics and background of the review process, code reviewer recommendation and multi-objective optimization.

Chapter 3 gives a review of the literature in detail and the significant work done by researchers in the past few years. The systematic literature review is composed of three main sections. The first section is the review protocol which gives details on the methodology using which the literature review is carried out. Section two offers details on research works carried out in this area in form of research questions and tables. Whereas, section three highlights the research gaps that were encountered.

Chapter 4 consists of the proposed approach in detail. It discusses the approach in terms of an overview of the algorithm (NSGA-III, fitness functions), main components of the approach and solution representation.

Chapter 5 includes implementation, validation and discussion of results accompanied by research question and relevant figures. It also brings detail to the comparison of our work with the state of the art. Additionally, it briefly explains the limitations of our work.

Chapter 6 concludes the thesis and reveals the future scope of this research.

The thesis outline is shown in Figure 1.

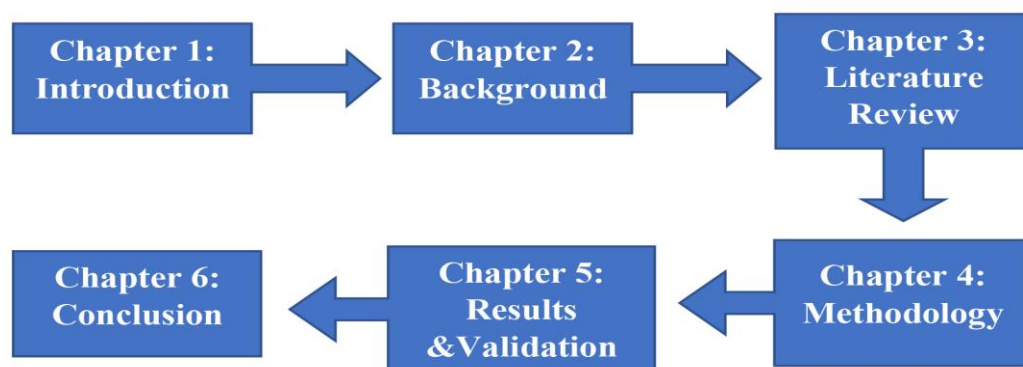


Figure 1. Thesis Outline

CODE REVIEWER RECOMMENDATION

Chapter 2 : Code Reviewer Recommendation

In this section, we first describe the necessary background related to the review process, code reviewer recommendation (CRR) and multi-objective optimization.

2.1 Review Process & CRR

In the last ten years, a wide range of research has been published in renowned databases in the context of Code Reviewer Recommendation (CRR). A code review process is an interaction between the submitter and reviewer/s. A series of meetings between the reviewer and the submitter are taken place, to ensure a mutual understanding of the change made and the review feedback. The changes in the code are performed by the programmer that is also the owner of the code and then submits it for the review request. The reviewers add comments as feedback about the new changes suggested by them [11], [12].

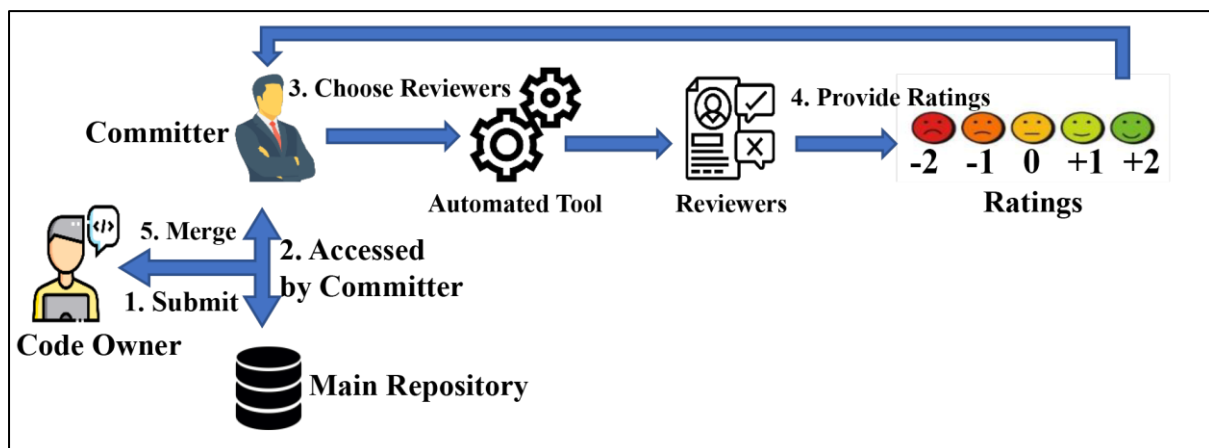


Figure 2. The MCR Process [5]

The process of MCR has been enunciated in Figure 2. This process begins when a code file/patch is placed in a certain repository by the code owner and a formal request for review is initiated to the committer [13]. Code-level change is assigned to each new file/commit. The Committer thereafter selects a suitable reviewer using an automated tool. The reviewers review the files and rate them as per the change effort. Rating is done using a standardized scale i.e., from -2 ~ +2 [14]. If a reviewer considers the file/patch perfectly fine i.e., if it does not require any change, the specific file/patch is rated as +2 and returned to the committer. The committer then merges the file/patch into the main project. If a file/patch is assigned the rating of -2, which means that a major change is required in the file/patch. The committer returns it to the owner, who submits it again after the suggested changes as per the review. Any other rating within the scale reflects the efforts required. The branch is merged into main

branch only if the collaborators accept those changes with +2 ratings [15]. Here, it may be safely claimed that the process of MCR (which contributes toward the overall quality of the project) solely depends on the appropriate selection of a reviewer. If done in an automated manner based upon appropriate logic, surely the outcome would be a quality review. MCR process is supported these days by many tools i.e., Gerrit (<https://www.gerritcodereview.com/about.html>).

The screenshot shows a Gerrit code review interface. At the top, it indicates the review is 'Active' and provides the ID '135008' and the title 'tdf#149252 update help for Entire Paragraph Area positioning'. Below this, the 'Change Info' section includes: 'Updated' at 16:31, 'Owner' Seth Chaiklin, and 'Reviewers' Ming Hua, Rafael Lima, Mike Kaganski, and Jenkins. The 'CC' section lists 'Repo | Branch help | master', 'Parent a611229', 'Topic' (with a pencil icon), 'Strategy Cherry Pick', and 'Hashtags'. The 'Submit requirements' section shows 'Code-Review' (No votes) and 'Verified' (+1 Jenkins). The main content area shows the commit message: 'tdf#149252 update help for Entire Paragraph Area positioning' followed by a detailed explanation of the changes to 'Paragraph Area' and 'Margin' to 'Entire Paragraph Area' for both Horizontal and Vertical positions. At the bottom, it shows 'Comments' with '2 unresolved' and '7 resolved'.

Figure 3. Example of code review (LibreOffice)

A detailed example of code review i.e., an open pull request at LibreOffice, (<https://gerrit.libreoffice.org/c/help/+/135008>) where many possible reviewers can be assigned to review the one change is shown in Figure 3. It also shows the details such as owner name, last updated status, rating marked by the respective reviewer and the topic of the code review.

2.2 Multi-objective Optimization

Multi-objective optimization is defined as “to find the trade-off balanced optimization between more than one objective”. Usually, it is difficult to find such a solution that provides multi-objective optimization because the objectives in the problem are mostly conflicting. To find and propose such solutions, Genetic Algorithms are used. These algorithms provide more than one solution also known as alternate possible solutions. The goal is to produce solutions that can optimally satisfy each objective concurrently. These algorithms consider certain credentials i.e., reviewer’s profile, experience, workload, expertise and commitment status, etc. [16]. These credentials are termed ‘Objectives’ and an algorithm may use a single

objective, may incorporate multiple objectives, or may utilize many objectives to suggest a reviewer [9], [17]. For example, an algorithm considering only one objective (i.e., experience) may be termed a single objective utilization algorithm [18]. The algorithm that takes into account two or three objectives is termed a multi-objective optimization algorithm [19] and an algorithm taking into account more than three objectives, while deciding on a reviewer, is termed a many-objective algorithm [9].

A random population of random pop size known as individuals is initiated at the beginning of Genetic Algorithms. The target problem is defined and then evaluated by using a fitness function (one fitness function for one respective objective) for the population. A random selection of 2 chromosomes as parents is selected to perform crossover and mutation operators on them. While applying the selection operator, it is ensured to select the individuals with the highest fitness values so that the new individuals come up with a higher probability to undergo crossover and mutation operators. A new population is created from an old population with the purpose to keep producing a better population every next time. The stopping criteria are defined to stop the algorithm where the individuals/population found is the best and fittest population [20], [21]. The flowchart of a genetic algorithm is displayed in Figure 4.

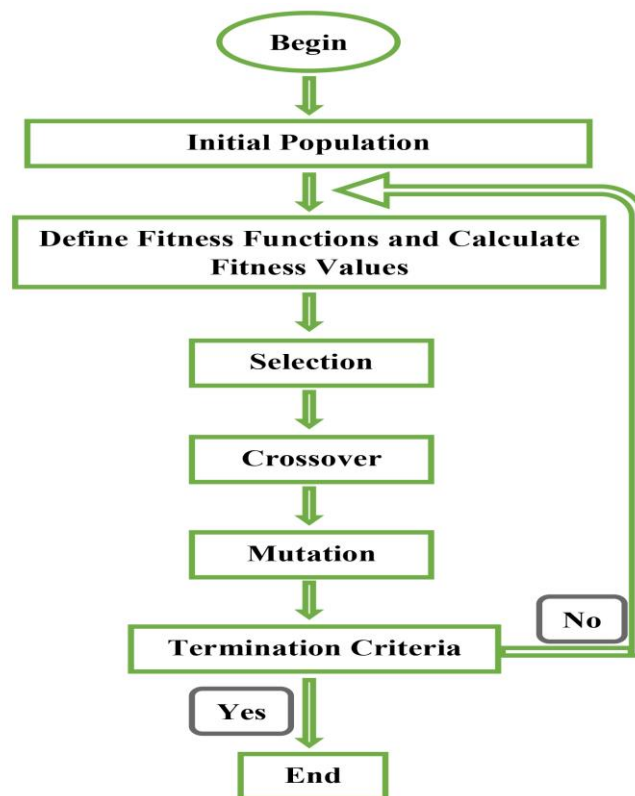


Figure 4. Genetic Algorithm Flowchart

Chapter 3

LITERATURE REVIEW

Chapter 3 : LITERATURE REVIEW

This chapter contains the systematic literature review performed for our research. It includes overview and major outcomes of SLR + contributions of SLR, review methodology, research questions, category definitions, review protocol of literature review, results and analysis, answers to the research questions for literature and conclusion of SLR.

3.1 Overview and Major outcomes of SLR

In Literature, there exist several studies (amply highlighted throughout this SLR), where various approaches, algorithms, tools/ techniques, and validation of various tools are highlighted in bits and pieces. Whereas, comprehensive research covering all the major aspects concerning the automated selection of a reviewer, as part of ‘MCR’, is hard to find in literature.

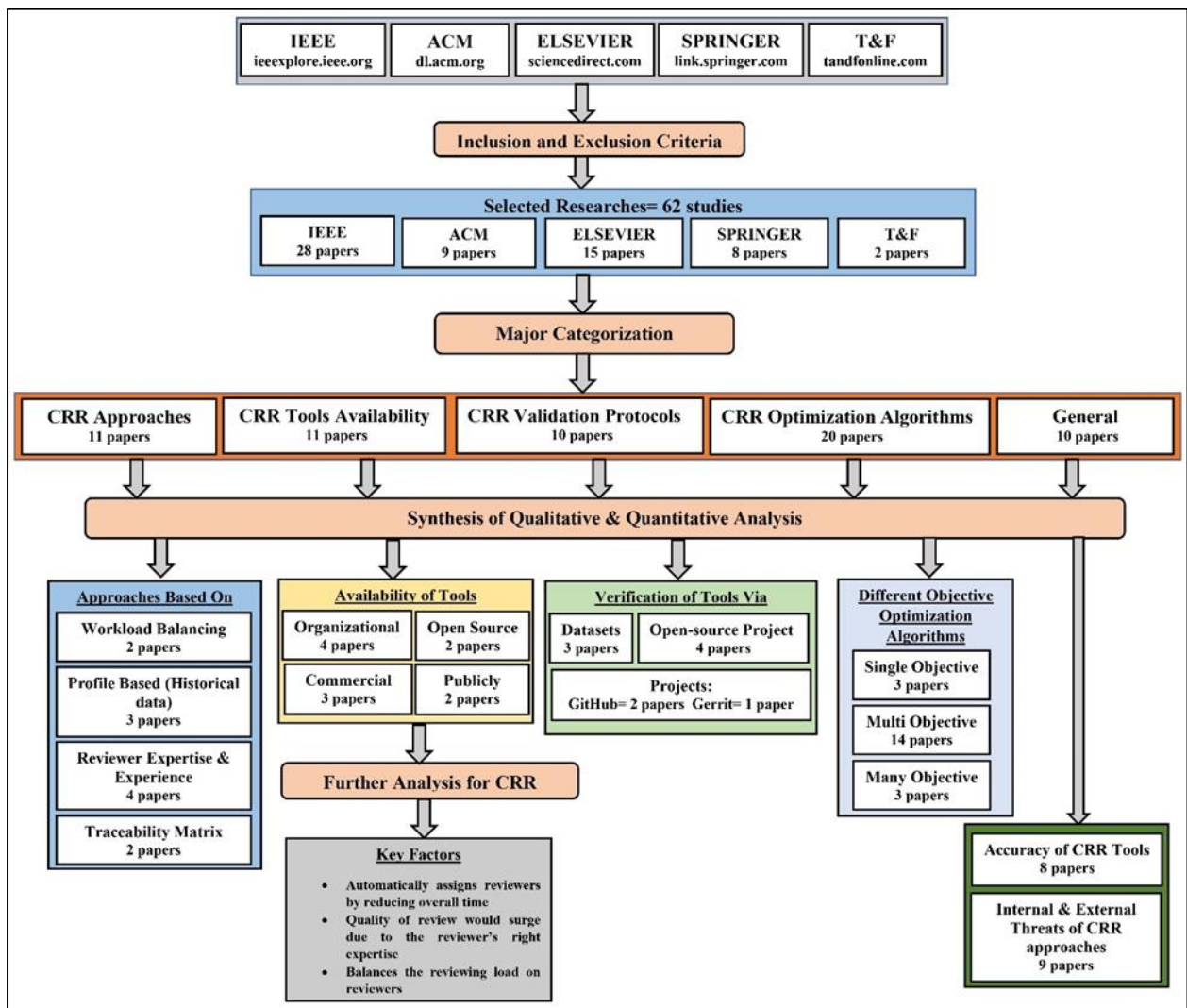


Figure 5. Overview & Major Outcomes of SLR

Overview/ major outcomes of this SLR are summarized in Figure 5. 62 landmark studies (published from 2011 – to March 2022) in renowned databases were investigated and analyzed after filtering them out via selection criteria. For ease of analysis and to explore the studies as per the research questions, studies were further categorized into five groups i.e., CRR Approaches (11 studies), CRR Tools Availability (11 studies), CRR Validation Protocols (10 studies), CRR Optimization Algorithms (20 studies) and a general category (10 studies). To obtain the required and precise results, a combination of qualitative and quantitative analysis is performed on the selected studies. As highlighted in Fig. 5, there are four types of approaches found in the literature for code reviewer recommendation i.e., ‘Load balancing’ (2 studies), ‘Profile-based’ (3 studies), ‘Reviewer expertise/experience’ (4 studies) and ‘Traceability matrix’ (2 studies). 14 tools were identified which are currently being used for automated CRR. The availability of tools is also classified into ‘Organizational’ (4 studies), ‘Open-source’ (2 studies), ‘Commercial’ (3 studies) and ‘Public’ (2 studies). The validation of tools is provided based on ‘Datasets’ of different sizes (3 studies), ‘Open-source projects’ (4 studies) and projects by ‘GitHub’ and ‘Gerrit’ (3 studies). Furthermore, optimization algorithms (in total 25 identified) are divided into 3 subcategories based on their type i.e., ‘Single objective’ (3 studies), ‘multi-objective’ (14 studies) and ‘Many-objective’ (3 studies). Based on a detailed analysis of results, key factors of CRR are identified and examined.

The **major contributions** of SLR are:

- a) Identifying various approaches to CRR
- b) Reporting various automated CRR tools
- c) Reporting the availability status of the tools
- d) Identifying the leading validation techniques/protocols through which the validity of various tools has been demonstrated by worthy researchers
- e) Enunciating various CRR algorithms and categorizing the identified algorithms as Single, Multi and Many Objective optimization algorithms based on the number of credentials taken into account while recommending a reviewer
- f) Identifying research productivity (frequency of use) of various optimization algorithms and tools
- g) Summarizing the external and internal threats to validity
- h) Ranking of CRR tools based on identified factors and accuracy

3.2 Review Methodology

Kitchenham guidelines [15] on SLR methodology are followed in this SLR. The major sections of this methodology are planning, conducting and reporting. In this methodology, Review Protocol is an integral step and is mainly divided into two categories that are Category Definition and Review Protocol Development. Briefly, this section deals with ‘Category Definition’ and ‘Review Protocol’. Moreover, this section implicates the research questions for review that are highlighted in Section 3.2.1.

3.2.1 Research Questions

Research questions have been summarized as below:

RQ1: What are the leading different Code Reviewer Recommendation tools defined from 2011- to March 2022?

RQ2: How the leading CRR tools are validated?

RQ3: What are the (Single, many and multi objectives) optimization algorithms identified in previous research?

RQ4: Which multi-objective optimization algorithm has the better research productivity over the years from 2011-March 2022?

RQ5: How the leading CRR tools may be ranked as per their accuracy?

RQ6: What are the threats to the validity of the previous finding in context to code reviewer recommendation tools?

3.2.2 Category Definition

To simplify the data extraction and synthesis process, we have defined five categories as shown in Table 1. The reviewed research articles are categorized later on into these categories. These categories are CRR Approaches, CRR Tools Availability, CRR Validation Protocols, CRR Optimization Algorithms and General category.

Table 1. Categorization of Review Information

Sr. #	Category	Types	Extracted Information			
1	CRR Approaches	4 types of Approaches	Identified CRR Approaches			
			Load Balancing (2 studies)	Reviewer Profile-based (3 studies)	Expertise and Experience (4 studies)	Traceability Matrix (2 Studies)
2	CRR Tools Availability	4 types of Availability	Identified CRR Tools Availability			
			Organizational (4 studies)	Open Source (2 studies)	Commercial (3 studies)	Publicly available (2 studies)

3	CRR Validation Protocols	4 Protocols	Identified CRR Validation Protocols			
			Datasets studies) (3)	Open-source projects (4 studies)	GitHub project (2 studies)	Gerrit Project (1 study)
4	CRR Optimization Algorithms	3 types of Algorithms	Identified CRR Optimization Algorithms			
			Single Objective (3 papers)	Multi-Objective (14 papers)	Many Objective (3 papers)	
5	General	It is highlighted that a specific paper may fall into one or more of the above-mentioned categories depending upon the availability of concerned information. For example, paper [13] falls under ‘CRR Approaches’, ‘CRR Optimization Algorithm’ and ‘General’ categories.				

- a) CRR Approaches:** From the reviewed literature, it was identified that four major approaches are being followed to recommend a reviewer for a specific piece of code. We have categorized the reviewed research articles into these categories which are ‘Load Balancing’ (2 studies), ‘Reviewer Profile-based’ (3 studies), ‘Reviewer Expertise and Experience’ (4 studies) and ‘Traceability Matrix’ (2 studies). Selecting the right approach for CRR is a critical task to perform while doing code reviews. For example, the authors of [16] have recommended the reviewers based on their expertise and experience while working on cross-project and specialized technologies. Additionally, the authors of [17] have presented an approach i.e., named ‘ADCR’ to recommend reviewers based on the reviewer’s profile-based historical data.
- b) CRR Tools Availability:** From the reviewed literature, 14 tools are identified that are used for CRR. The availability of these tools is a critical question. Under this heading, we have sub-categorized (4 sub-categories) the studies based on the availability of their recommended tools as shown in Fig. 1. Some tools for CRR are ‘Open Source’ with their source code available as well. Other types of availability are ‘Public’, ‘Commercial’ and ‘Organizational’. For example, Tie [18] is a tool based on text mining and a profile location-based approach and its availability is organizational.
- c) CRR Validation Protocols:** Another important aspect is the protocol that is used for the validation of CRR tools. The reviewed literature highlighted that there are various techniques used for the validation of CRR tools to claim their accuracy. These protocols are used to perform validation on different datasets, GitHub and Gerrit projects and open-source projects. Under this category, research studies are sub-categorized based on the CRR tools validation protocols as shown in Table 7. For example, WhoReview [19] tool was validated by authors on four big open-source projects in Microsoft.

- d) CRR Optimization Algorithms:** A review of the selected literature revealed that there exist various optimization algorithms. These algorithms are classified as single, multi and many objectives optimization algorithms. For example, a single-objective (i.e., availability of reviewer) optimization algorithm has been proposed by the authors of [20]. Likewise, the authors of [21] presented a multi-objective optimization algorithm that takes into account three objectives i.e., availability, expertise and collaboration of reviewers to recommend a reviewer. Moreover, some studies have presented many-objective algorithms. Collectively, all such studies that provide algorithms for single, multi and many-objective algorithms are placed under this CRR Optimization Algorithms.
- e) General:** This category contains those research studies that can be a part of more than one above-mentioned category. For example, papers [22], and [23] have used the NSGA-II optimization algorithm with 2 different tools. They have also mentioned the validation techniques via two datasets respectively, so they are placed under the ‘CRR Validation Protocols’, ‘CRR Optimization Algorithm’ and ‘General’ category.

3.2.3 Review Protocol

Once the categories are defined, the Review Protocol is shaped as per the defined methodology of Kitchenham [22]. Review protocol consists of standard six steps. The first two steps (background and research questions) are already elaborated while the remaining steps are explained in the undermentioned headings:

I. Acceptance and Rejection Criteria

Acceptance and rejection criteria primarily contain the set of rules/criteria that form the basis of acceptance/rejection or inclusion/exclusion of a specific paper. The criteria consist of some steps that need to be followed for deciding whether the study is excluded or included. Studies that do not follow or fulfill the following parameters are not considered. Only those studies are considered for further scrutiny that passes through the initial filter of acceptance and rejection criteria.

- a) Subject:** Only those papers should be selected that are completely relevant to code reviewer recommendation. Studies dealing with code reviews [3] but discussing code reviewer recommendations marginally should be discarded. As this SLR deals solely with Code Reviewer Recommendation therefore those papers should be discarded that don't contribute positively towards improvement in the automated CRR.

- b) **Publication Year:** This SLR deals with those studies that are published from January 2011~ to date. Studies published before 2011 were rejected. Primarily, the latest research are based on the findings of previous research. Therefore, a very carefully considered research duration is considered for this SLR. This duration not only encompasses the latest trends related to the domain of CRR but it also covers the previous researches logically as well. For example, paper [8] discusses the latest multi-objective optimization algorithm, however, in actuality, the concept of multi-objective optimization algorithms started back in the early 2000. So, this paper [8] amply covers the range from 2000 ~ to 2016.
- c) **Publisher:** The studies were selected from numerous renowned and well-known scientific repositories. The databases of IEEE, ELSEVIER, ACM, SPRINGER and Taylor & Francis are utilized to conduct this SLR. These databases are reliable and trustworthy and studies that are published in these databases undergo a stringent peer review. Therefore, studies considered for this SLR must fall under these databases.
- d) **Language:** To conduct this SLR, the studies that are published in English are considered. Those studies which are written in a language other than English are excluded. We found 3-4 relevant papers with an abstract in the English language, but the paper content was not in English.
- e) **Validation:** Only those studies should be included, where validation of CRR tools is performed properly either with the help of a dataset, open-source project (Gerrit, GitHub & Microsoft), or with the help of a case study. For example, a study [23] proposed a code reviewer recommendation tool ‘RSTrace+’ with a complete validation via an open-source project and accomplished top-3 recall & precision values. Subsequently, studies that have lacked, insufficient or missing validation techniques are excluded from this SLR.

II. Search Process

After specifying the acceptance and rejection criteria, the search process is initiated by exploring the renowned databases of IEEE, ELSEVIER, ACM, T&F and SPRINGER. We used different search terms or keywords to perform this process as shown in Table 2. We started the search process with the most relevant search words like “Code Reviewer Recommendation”. These search terms resulted in hundreds of results, which could not be fully examined. For example, the Springer database resulted in 23759 contents for the search term “Code reviewer recommendation” in default settings.

Table 2. Search Keywords and Results

Sr. #	Search Keywords	Operator (AND/OR)	No. of Search Results				
			IEEE	ACM	T&F	Springer	Elsevier
1	Code reviewer recommendation	AND	33	8	2	7	3
		OR	81	30	9	31	24
2	Tools for code reviewer recommendation	AND	24	0	0	8	0
		OR	50	12	4	19	9
3	Code optimization algorithms	AND	32	15	7	5	8
		OR	80	52	16	34	43
4	Types of optimization algorithms	AND	0	4	1	7	0
		OR	115	14	5	19	10
5	Genetic algorithms	AND	43	32	9	80	20
6	Threats to validity of CRR	AND	25	0	2	6	0
		OR	49	23	7	30	19
7	CRR, Genetic algorithm techniques	N/A	90	39	12	98	4
8	Validation of CRR tools	N/A	21	14	8	13	9

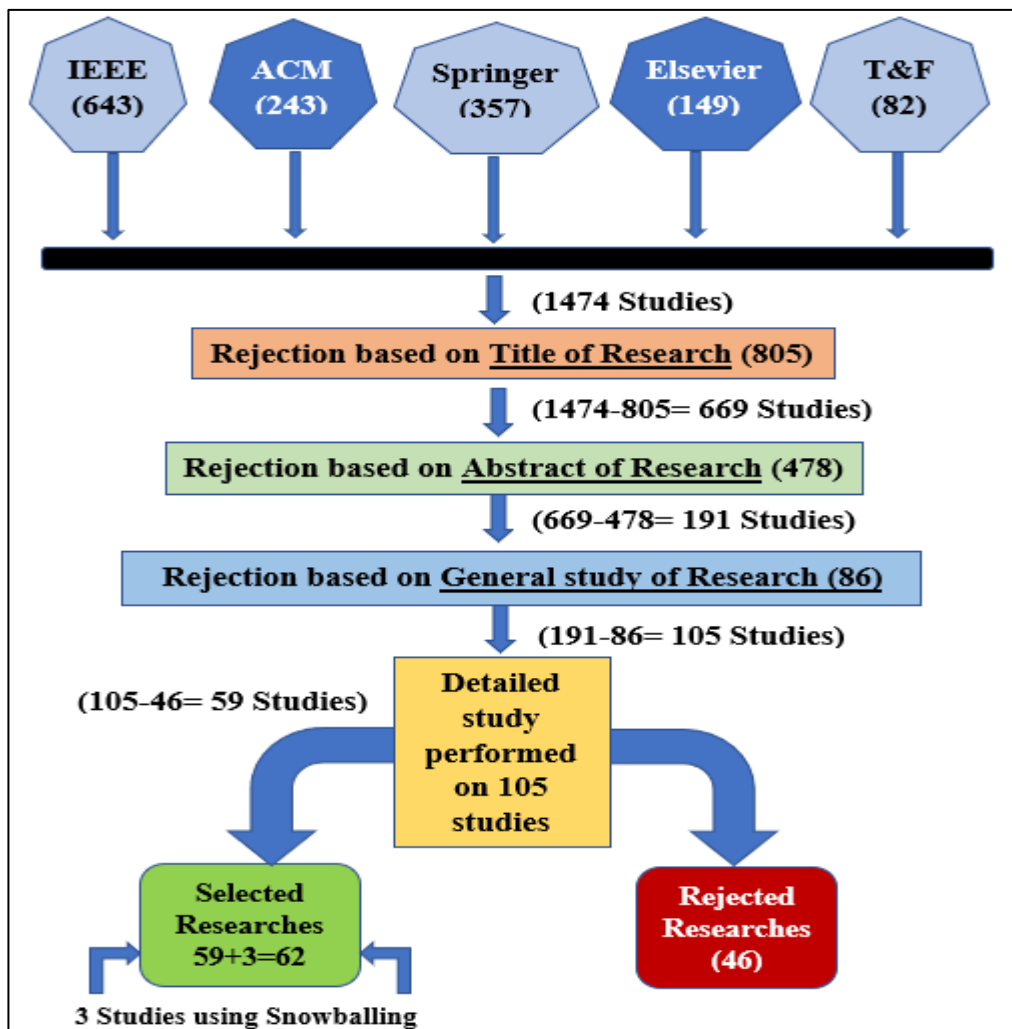


Figure 6. Search Process

To refine our results, we applied some filters, like we put the range (2011-March 2022) on publication years while searching for related research work. Two kinds of operators (AND, OR) are used to carry out the search process via keyword searching. Moreover, we tried to identify the synonyms and possible substitutes for the terms to make the search process in detail. We used the Snowballing search method [25]. The ‘forward snowballing’ (to check where that paper was referred/ cited) and backward snowballing (what citations that paper used) to identify related additional papers. After applying all these filters and detailed search options, we were able to gather the finest and most relevant results that could be easily studied and beneficial for this SLR. For example, 114 studies were figured out for further study with the search term “Code Reviewer Recommendation” after applying above mentioned techniques. Similarly, after applying the same techniques, the search term ‘Genetic Algorithm’ in the ACM database only resulted in 32 papers. In this manner, after a comprehensive search process, we finally obtained 62 research studies to get accurate and reliable answers to our research questions. The complete search process is elaborated in Fig. 6 and the steps are explained below:

- Initially, 1474 research studies were obtained from all the databases. Firstly, we analyzed the titles of research to check the relevance according to inclusion and exclusion criteria. We rejected 805 because those studies indicated their irrelevance in their title.
- Then, we studied the abstracts of the remaining 669 papers. It was observed from a few studies that their abstract violates the parameters defined in inclusion and exclusion criteria. Thus, we discarded 478 studies by examining their abstracts.
- We investigated the remaining 191 research studies. Until here, the validation criteria were not considered and only the first four acceptance and rejection parameters were followed. Hereon, the 5th parameter i.e., validation was also considered by studying various sections in detail. Therefore, based on this investigation via general study and then detailed in-depth study, we excluded 86+46 research studies respectively and included 59 studies, which were conforming completely to our selection and rejection criteria.
- Finally, we applied snowballing process. This process ensured the selection of relevant studies that have been missed by chance in the search process. We utilized both forward and backward snowballing. As result, 3 studies were selected that appeared relevant to our research context by the snowballing process.

Table 3. Accepted Research Papers Vis a Vis Databases

Sr. #	Scientific Catalogue	Sort	References for Studies (selected)	Total Researches
1	IEEE	Conference	[11], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42]	28
		Journal	[43], [44], [45], [46], [47], [48], [49], [50]	
2	Elsevier	Journal	[7], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64]	15
3	Springer	Conference	[65]	8
		Journal	[9], [66], [67], [68], [69], [70], [71],	
4	ACM	Conference	[23], [72], [73], [74], [75], [76], [77], [78]	9
		Journal	[79]	
5	Taylor & Francis	Journal	[80], [81]	2

III. Quality Evaluation

We ensured to select research from well-known and high-impact studies (authentic and accepted worldwide) from scientific databases to guarantee the reliable results/ conclusions of this SLR. 5 most renowned databases are used for the selection of studies according to parameter no 3 (Publisher) as enunciated in the inclusion and exclusion criteria. Table 3 presents the detailed distribution of selected studies concerning publishing databases. This table elaborates the information about the names of the databases, selected studies for referencing, their sorting based on publication type (journal or conference) and the total count of studies concerning the database. 28 studies are selected from IEEE, 15 from Elsevier, 8 from Springer, 9 from ACM and 2 are selected from Taylor & Francis.

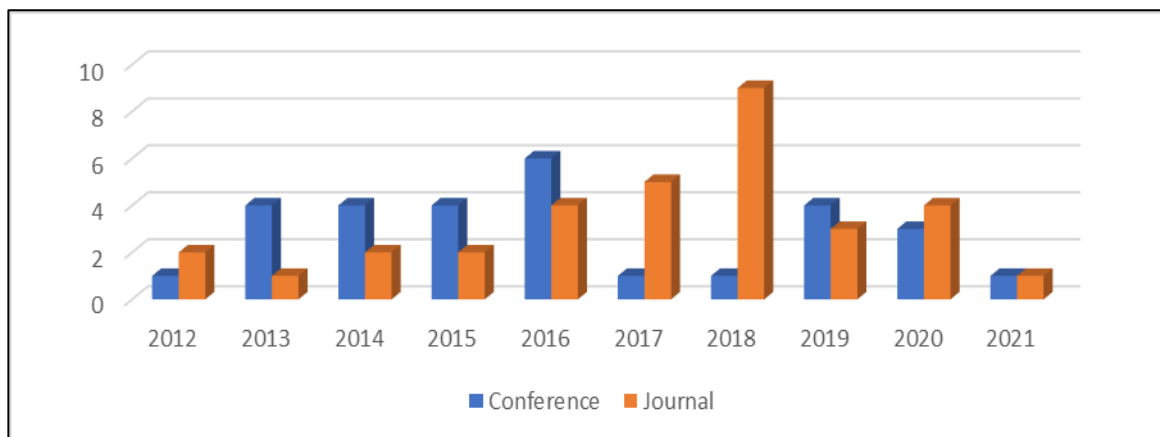


Figure 7. Distribution of Selected Studies based on Publication Year

Furthermore, within a specific database, the distribution of conference and Journal publications can also be seen. For example, there are 8 journals and 20 conference papers that have been selected from the IEEE database Elsevier database contributed 15 Journals. 1 conference paper and 7 journals are selected from Springer, and 2 journal papers are contributed by Taylor & Francis. Finally, from the ACM database, 8 conferences and 1 Journal were added to the list of papers that were scrutinized for the sake of this SLR.

Table 4. Year-wise distribution of Selected Studies

Sr. #	Year	Studies	Contribution percentagewise	Total
1	2012	[27], [48], [59]	4.84%	3
2	2013	[29], [34], [37], [41], [63]	8.06%	5
3	2014	[40], [42], [44], [45], [65], [78]	9.68%	6
4	2015	[11], [30], [35], [39], [68], [79]	9.68%	6
5	2016	[7], [24], [25], [28], [36], [46], [61], [75], [76], [80]	16.13%	10
6	2017	[43], [31], [47], [49], [52], [55]	9.68%	6
7	2018	[33], [54], [57], [58], [62], [64], [67], [69], [70], [71]	16.13%	10
8	2019	[23], [32], [38], [50], [56], [60], [72]	11.29%	7
9	2020	[26], [9], [53], [66], [73], [77], [81]	11.29%	7
10	2021	[51], [74]	3.23%	2

The horizontal axis of Fig. 7 shows the year of publication and the publication type. Whereas, the vertical axis shows the number of publications in a particular year. The tabular form of Fig. 7 is shown in Table 4 which presents the year-wise distribution of selected research studies. As it can be seen that this SLR encompasses a blend of the latest information from the most reliable resources. The last column (Total) gives the total number of publications in a particular year. The same table gives the percentage-wise distribution of research studies per year. As it may be noticed that the number of studies from 2016~2020 is contributing more than in other years. It may be safely concluded that the conclusions of this SLR encompass the latest information. Similarly, the databases of IEEE and Elsevier contributed to a larger number of studies than other databases. In comparison, lesser studies were extracted from T&F, Springer and ACM. Resultantly, the total count of papers (under consideration) comes out to be 62 studies.

The sorting of publication types (journal and conference) is also a very important factor to demonstrate the quality of this SLR. We included 33 Journal papers out of 62, which means a total of 53% contribution of Journals in this SLR. Likewise, 29 conference studies were found to be about 47% contribution. All of these publications fully qualify the inclusion and exclusion criteria set for this SLR.

IV. Data Extraction and Synthesis

After selecting studies according to inclusion and exclusion criteria, we have developed a template to complete the data extraction and synthesis step as shown in Table 5. The answers to research questions are systematically found by doing this step. Moreover, it also helped to gather and analyze the concerning fundamentals from chosen studies. The data which was extracted from the selected research studies include Bibliographic information, generalized overview, proposed methodology, implementation details, outcomes, limitations, categorization, tools and validation techniques discussed in a particular study. In this manner, answers to all the research questions were categorically found. The template helped invaluablely in sifting the essentials from the non-essentials.

Table 5. Template of Data Extraction and Synthesis

Sr. #	Type	Details
1	Bibliographic information	It includes title, publication details (year, authors information) and type of research paper (Conference or Journal)
2	Summary	The proposal of each selected study is analyzed
3	Proposed Methodology	It contains the methodology followed by each study
4	Implementation Details	It includes the details of the technology that is used to implement the proposed methodology
5	Outcomes	We analyzed the outcomes of each study thoroughly
6	Limitations	Limitations while achieving any objective (if any)
7	Categorization	Selected studies are classified as per defined categories. the results are summarized in Table 1
8	Tools	Well-known Tools are analyzed and summarized in tables (Table 6)
9	Validation	Validation methods of each study e.g., datasets, and open-source projects (Table 7)
10	Algorithms	Algorithms are proposed by each research study to achieve the required objective. Results are summarized in Table 8

3.3 Results, Analysis & Answers to Research Questions

The primary aim of this research article is to examine and query the selected literature to reliably answer the identified research questions. This section reports the results after careful examination of various aspects of extracted data. It is pertinent to mention that the Journals generously contributing to our quest for the latest trends in CRR are; ‘Information and Software Technology’, ‘Applied Soft Computing’, ‘Journal of Intelligent Information Systems’, ‘Journal of Computer Science and Technology’, ‘Information Sciences’, ‘IEEE Transactions on Evolutionary Computation’, ‘Natural Computing’, ‘Automated Software Engineering’ and ‘IEEE Access’. Likewise, numerous worthy conferences i.e., ‘International Conference on Software Engineering Companion’, ‘International Conference on Software

Engineering’ (ICSE), ‘International Conference on Software Analysis, Evolution and Reengineering’ (SANER) and many others contributed to our SLR.

1) Leading CRR Tools:

One of our research questions queries the leading CRR tools. This section presents the latest CRR tools and their availability as reported by the worthy researchers in selected research. 14 of the most acclaimed tools that are used for reviewer recommendation are shown in Table 6. These tools are reported in 11 types of research published from the year 2011-March 2022. In this table, tools are presented along with their abbreviations (if any), availability and the concerned studies. 4 types of availability have been associated with the CRR tools i.e., Organizational or Proprietary, publicly available, Open source and Commercial. An Organizational or Proprietary tool is the one that legally remains the property of a specific organization, group or individual who owns it like, Trello is propriety to ‘Atlassian’. The publicly available tool is licensed at no cost or for an optional fee e.g., SQLite and Review Bot. The source code for this kind of tool is kept hidden from the public. The tool that is freely available and its source code is published freely as well to use or modify falls under the Open-source category i.e., CORRECT. Lastly, the commercial tool may be owned (via authorized licensing) by anyone who pays for it e.g., RevRec.

After analysis of the tools, it may be safely claimed that tools i.e., CORRECT and RSTrace+ (Reviewer suggestion by Traceability graph) are open sources. Carrot, RevFinder (Review finder) and Review Bot are publicly available. The tools that are commercially available for reviewer recommendation are WhoDo, Profile-based, WhoReview, Latent Dirichlet Allocation and RevRec. Similarly, tools like TIE (Text mining and file location), ADCR (Appropriate Developer for Code Review) and CoreDevRec are organizational tools.

Table 6. Leading Code Reviewer Recommendation Tools

Sr. #	Tools	Abbreviation (if available)	Availability	Reference
1	TIE	Text mining and file location	Organizational	[30]
2	ADCR	Appropriate developer for Code Review		[26]
3	CoreDevRec	Core Member Recommendation for Evaluation		[52]
4	Naïve Bayes	N/A		[33]
5	CORRECT	N/A	Open Source	[25]
6	RSTrace+	Reviewer suggestion by Traceability graph		[53]
7	RevFinder	Review Finder	Publicly Available	[33]
8	Carrot	N/A		[26]
9	Review Bot	N/A		[29]
10	RevRec	Review recommendation	Commercial	[28]
11	WhoDo	N/A		[24]

12	Profile Based	Profile of Individual Recommender	[32]
13	WhoReview	N/A	[51]
14	LDA	Latent Dirichlet Allocation	[23]

2) Leading Validation Techniques:

Various techniques are in vogue to demonstrate the validity of tools. It has been noticed that various researchers have used open-source projects, datasets, Gerrit and GitHub projects for validity. For example, Z. Liao et al. [60] proposed validation via dataset. In a research article [27], M. M. Rahman et al. presented a validation approach i.e., based on the Gerrit repository project. Similarly, C. Yang et al. [63] identified a validation approach based on GitHub projects. Therefore, in this section, the effort has been made to figure out the validation techniques that are used by worthy researchers to validate the leading CRR Tools. In Table 7, validation techniques are mentioned against the leading CRR tools.

We found 3 studies [12], [16] and [60] where validation of 5 tools is reported via datasets. i.e., LDA, Review Bot, TIE, Naïve Bayes and Rev Finder. Similarly, Tools i.e., CORRECT, Carrot, Profile-based, Who Review and RSTrace+ are validated via Open-source Projects as reported by [67], [68], [69], [70]. Likely, Who Do, CoreDevRec and RevRec are validated via GitHub Repository projects [62], [63]. Lastly, ADCR has been validated via the Gerrit repository project named ‘Core-Plugins’ [27].

For elaboration, we discuss the ‘CORRECT’ tool. It has been validated via 10 open-source projects. The data in these open-source projects were about Atomix, Tablesaw, Vavr, Takes, Dkpro-core, Pac4j, Open stack, Android and Qt. This tool successfully served 17115 pull requests (project files submitted for reviewer recommendation) concerning these open-source projects.

Table 7. Leading CRR Validation Protocols

Sr. #	Tools	Validation Techniques	Reference
1	CORRECT	10 open-source projects (17115 pull requests)	[76]
2	WhoDo	Five repositories of GitHub	[72]
3	Carrot	Open-source Ericsson Project	[73]
4	Review Bot	Two Datasets (one large and one small)	[60]
5	TIE	Two Android pull requests datasets	[12]
6	ADCR	Gerrit code review project	[31]
7	Profile Based	Four large open-source projects	[75]
8	WhoReview	Four long-lived open-source projects in a large organization	
9	CoreDevRec	Five popular projects with 18651 pull requests in GitHub	[68]
10	Naïve Bayes	Dataset of 51 projects	[16]
11	RevFinder	Dataset of 51 projects having 42,045 reviews	
12	LDA	Two Android pull requests datasets	[12]
13	RSTrace+	Evaluated on 40 open-source projects	[74]

14	RevRec	Two popular projects in GitHub	[69]
----	--------	--------------------------------	------

3) Optimization Algorithms:

Primarily, CRR tools are based on certain optimization algorithms. These algorithms serve as the backbone of these tools. A tool works on a specific algorithm to recommend a reviewer. A specific optimization algorithm takes into account certain objectives e.g., ‘reviewer experience’, ‘reviewer workload’ and ‘reviewer ~ developer collaboration’. A review of the selected literature revealed that there exist various optimization algorithms. To answer research question 3, We have identified the 25 well-known, recently used and distinguished optimization algorithms from 2011-March 2022 as shown in Table 8, which explains the algorithm, its abbreviation and its type i.e., single, multi and many. For example, a single-objective (i.e., availability of reviewer) optimization algorithm has been proposed by the authors of [20]. Likewise, the authors of [21] presented a multi objectives optimization algorithm that takes into account three objectives i.e., availability, expertise and collaboration of reviewers to recommend a reviewer. Moreover, some studies have presented many objective (more than three objectives) algorithms as well.

As noticed, the algorithms such as ‘Strength Pareto Evolutionary Algorithm’, ‘Niched Pareto Genetic Algorithm’ and ‘Pareto Archived Evolution Strategy’ are single-objective optimization algorithms. Similarly, ‘Non-dominated Sorting Genetic Algorithm’, ‘Multi-Objective Ant Colony Optimization’ and ‘Multi-Objective Genetic Algorithm’ are multi-objective optimization algorithms. ‘Decomposition-based Elitist Non-Dominated Sorting Algorithm’ (NSGA-III), ‘Unified Evolutionary Algorithm for Decomposition based Elitist Non-Dominated sorting’ and ‘ Θ -Dominance Sorting Genetic Algorithm’ are many objectives’ optimization algorithms. The most used and famous algorithm in researches is NSGA-II. Different variants of NSGA-II are introduced by different researchers lately e.g., ‘Multi-Objective Particle Swarm Optimization’ and ‘Non-dominated sorting and local search’ as reported by [59], [71] respectively.

Table 8. Optimization Algorithms

Sr. #	Algorithm	Abbreviation (if available)	Type	Reference
1	SPEA	Strength Pareto Evolutionary Algorithm	Single	[49]
2	NPGA	Niched Pareto Genetic Algorithm		[43]
3	PAES	The Pareto archived evolution strategy		[27]
4	SPEA2	Strength Pareto Evolutionary Algorithm 2	Multi	[59]
5	NSGA-II	Non-dominated Sorting Genetic Algorithm		[77]
6	IBEA	Indicator-Based Selection in Multi-Objective Search		[54]
7	MOPSO	Multi-Objective Particle Swarm Optimization		[59]

8	MOACO	Multi-Objective Ant Colony Optimization		[40]
9	SA	Simulated annealing algorithm		[58]
10	MOEA	Multi-Objective evolutionary algorithm		[55]
11	ECGA	The Extended Compact Genetic Algorithm		[51]
12	SEMO	The Simple Evolutionary Multi-Objective Optimizer		[70]
13	EMOA	Evolutionary Multi-Objective Algorithm		[33]
14	SMS-EMOA	S-metric selection Evolutionary Multi-Objective Algorithm		[79]
15	ε-MOEA	The ε-Based Multi-Objective Evolutionary Algorithm		[36]
16	PSEA-II	The Pareto envelope-based selection Algorithm II		[40]
17	MOGA	Multi-Objective Genetic Algorithm		[56]
18	C-MOGA	Cellular Multi-Objective Genetic Algorithm		[51]
19	MSOPS-II	Multiple single objective Pareto sampling algorithm II		[35]
20	MOEA/IGD-NS	A multi-objective evolutionary algorithm based on An Enhanced inverted generational distance metric		[49]
21	BCE-IBEA	Bi-criterion evolution for Indicator-Based Evolutionary Algorithm		[48]
22	NLSLS	Non-dominated sorting and local search		[71]
23	NSGA-III	Decomposition based Elitist non dominated sorting Algorithm	Multi + Many	[44]
24	Θ-NSGA-III	Θ-Dominance Sorting Genetic Algorithm	Many	[78]
25	U-NSGA-III	Unified Evolutionary Algorithm for Decomposition based Elitist non dominated sorting Algorithm	Single + Multi + Many	[65]

4) Validity Threats:

There may be two types of validity threats i.e., ‘Internal’ and ‘External’. Internal validity refers to how well the procedures of the study are validated or trustworthy while external validity relates to the transfer of results from one situation/event to another with similar characteristics. To answer our RQ6, there are some validity threats to the tools as well. To answer RQ6, we present Table 9, in which we have summarized the tools along with their internal and external threats.

To elaborate on the validity threats, we may discuss the ‘CORRECT’ tool (Ser 1 of Table 9). One of its internal threats is that it is validated only for medium-sized projects and its external threat is that it has only experimented with projects developed on Python, Java and Ruby specifically. Similarly, ‘ADCR’ (Ser 2 of Table 9) internal threat is that its validation gives results on Gerrit repository projects specifically. While its external threat is the Cold start problem, i.e., a situation where newly joined users are difficult to recommend as they don’t have any profile, experience, etc. So, external threat in the ‘ADCR’ tool is basically for newly joined code reviewers. The other external threat is the limitation of their tool result to 5

project datasets only. Table 9 shows other tools and their respective Internal and external threats respectively.

Note: Every research study provides mitigation ways/methods to overcome these threats in respective studies.

Table 9. Threats to Validity for CRR Tools

Sr. #	Tools	Threats to Validity		Reference
		Internal threats	External threats	
1	CORRECT	Validated on medium-sized subject systems only	Experimented on python, Java and Ruby projects	[76], [25]
2	ADCR	Validation gives results on Gerrit repository projects	Cold start problem for newly joined code reviewers	[26], [31]
3	WhoReview + TIE	Uses longitudinal data for evaluation only	Validated on provided dataset only	[8], [14]
4	CoreDevRec	Related to errors and bias in experimental data	Results are limited to 5 project datasets only	[62]
5	Naïve Bayes	Related to errors and bias in data (collection + implementation)	Results on only those 51 projects on GitHub	[17]
6	RevFinder	Involves manual examination in reviewing process	Limited to four dataset results only	[16]
7	LDA	Cannot ensure results validation in the initial phase of any project	Results may not be generalized on other open-source projects	[21]
8	RSTrace+	Related to data collection and implementation + Accuracy will be lower at the start of the recommendation		[23]
9	RevRec	Related to nonlinear or time-dependent validation results	Cannot generalize on other code review tools or projects	[24], [25]

5) Answers to Research Questions:

RQ1: What are the leading different Code Reviewer Recommendation tools defined from 2011-to March 2022?

Answer: In this SLR, we have identified 14 most acclaimed tools from selected studies that are used for reviewer recommendation as discussed and shown in Table 6. This table presents the names of the tools against their availability. It has been analyzed that the tools i.e., ‘CORRECT’ and ‘RSTrace+’ (Reviewer suggestion by Traceability graph) are open source. ‘Carrot’, ‘RevFinder’ (Review finder) and ‘Review Bot’ are publicly available. The tools that are commercially available for reviewer recommendation are ‘WhoDo’, ‘Profile-based’, ‘WhoReview’, ‘Latent Dirichlet Allocation’ and ‘RevRec’. Similarly, tools i.e., ‘TIE (Text

mining and file location), ‘ADCR’ (Appropriate Developer for Code Review), ‘Naïve Bayes’ and ‘CoreDevRec’ are organizational tools.

RQ2: How the leading CRR tools are validated?

Answer: Various validation techniques are utilized by the researchers to demonstrate the validity of their proposed tools. Table 7 presents the validation techniques for the identified CRR tools. The techniques are identified as ‘open-source projects’, ‘datasets’, ‘Gerrit’ and ‘GitHub’ projects. We found 3 studies [12], [16] and [60], where validation of 5 tools is demonstrated via ‘datasets’ i.e., ‘LDA’, ‘Review Bot’, ‘TIE’, ‘Naïve Bayes’ and ‘Rev Finder’. Similarly, Tools i.e., ‘CORRECT’, ‘Carrot’, ‘Profile based’, ‘Who Review’ and ‘RSTrace+’ are validated via ‘Open-Source Projects’ as reported by [67], [68], [69] and [70]. Likely, ‘Who Do’, ‘CoreDevRec’ and ‘RevRec’ are validated via ‘GitHub Repository Projects’ [62] and [63]. Lastly, ‘ADCR’ has been validated via the ‘Gerrit Repository Project’ named ‘Core-Plugins’ [27]. As may be noticed that most of the tools were validated with the help of ‘Open-Source Projects’.

RQ3: What are the (Single, many and multi objectives) optimization algorithms identified in previous research?

Answer: This SLR has identified the 25 well-known and distinguished algorithms (2011-March 2022) that are used by the CRR tools to recommend a suitable reviewer. Table 8 presents these algorithms. The type column consists of 3 categories: Single, multi and many. As noticed, the algorithms such as ‘Strength Pareto Evolutionary algorithm’, ‘Niche Pareto Genetic algorithm’ and ‘Pareto Archived Evolution Strategy’ are single-objective optimization algorithms. Similarly, ‘Non-dominated Sorting Genetic Algorithm’, ‘Multi-Objective Ant Colony Optimization’ and ‘Multi-Objective Genetic Algorithm’ are multi-objective optimization algorithms. ‘Decomposition-based Elitist Non-dominated Sorting Algorithm’ (NSGA-III), ‘Unified Evolutionary Algorithm for Decomposition based Elitist non-dominated sorting’ and ‘ Θ -Dominance Sorting Genetic Algorithm’ are many objectives’ optimization algorithms.

RQ4: Which multi-objective optimization algorithm has the better research productivity over the years from 2011-March 2022?

Answer: Research productivity is the frequency of use of a specific algorithm. After analysis of the selected research studies (2011~2022) and as may be noticed in Fig. 8, it may be safely claimed that NSGA-II [8], [9], [34], [55] and [71] has the highest research productivity. Similarly, it is the most widely used multi-objective optimization algorithm used for code reviewer recommendation lately. NSGA-II has gained popularity because of its non-dominant

sorting technique, which is very effective as it provides optimal solutions and also calculates each objective fitness function value separately [21] and [32]. Moreover, it provides the most reliable and steadfast recommendation in minimal conjunction time. NSGA-III [22], [38], [39], [54] and [72] is another type of algorithm from the NSGA family which takes into account ‘Many’ objectives. Different other algorithms in related researches are MOGA [58], [61], [64] and [75], IBEA [37], [41], [45] and [48], MOEA/D [38], [56], SPEA [44], [53] and MOACO [57]. Fig. 8 shows a bar chart between algorithms (vertical axis) and their usage frequency (horizontal axis) in research over the last ten years.

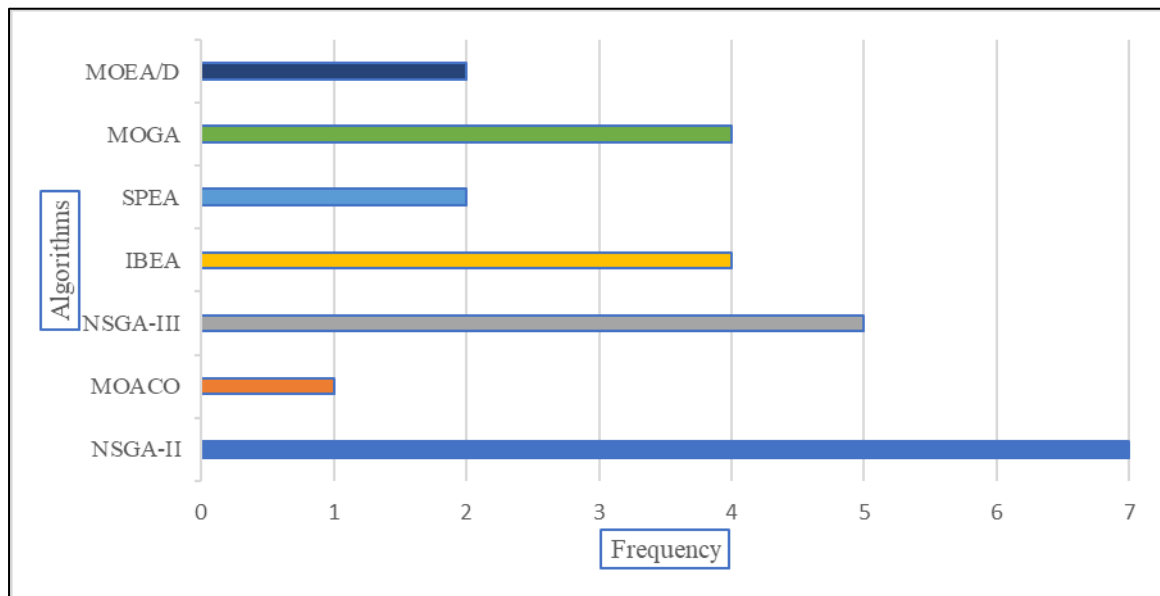


Figure 8. Research Productivity of Algorithms (2011~2022)

RQ5: How the leading CRR tools may be ranked as per their accuracy?

Answer: While answering this research question, an effort was put in to draw a comparison among accuracies of various identified CRR tools. After analyzing all the selected studies, we could only find 6 studies, where the accuracy of the reported tools was mentioned. These tools are mentioned in Table 10 along with the reported accuracies. It may be noticed that only three of the mentioned tools have used the same type of approach. So, in actuality, the real comparison can be drawn among these three tools i.e., ‘WhoReview’ (Ser 1 of Table 10), ‘TIE’ (Ser 2 of Table 10) and ‘Review Finder’ (Ser 3 of Table 10). The accuracy of the rest of the three tools along with the identified approach is also mentioned in this table.

From the review of the selected literature, ‘Who Review’ has been identified as the CRR tool with 85-92 % accuracy. Moreover, it takes into account the reviewers’ profiles while recommending. The accuracy of ‘TIE’ has been identified as 75-90 %, as it follows the reviewer profile and previous work done by reviewers. Likewise, ‘RSTrace+’ (Ser 4 of Table

10) recommends reviewers based on a traceability matrix approach with an accuracy of 80-85%. ‘TIE’ and ‘RSTrace+’ are almost on the same level of accuracy but they both use a different approach for a recommendation. The accuracy of the rest of the tools and their approach factors may be figured out in Table 10.

Table 10. Ranking of CRR Tools based on Accuracy

Sr.#	Tools	Identified Factor	Accuracy	Reference
1	Who Review	Recommend reviewers based on their reviewer profile	85-92%	[14]
2	Text mining and a file location-based approach (TIE)	Analyzes the reviewer profile and their similarity based on previous work done	75-90%	[8]
3	Review Finder	Recommend via reviewer profile similarity with the previously reviewed files	80%	[17], [19]
4	Reviewer Suggestion by Traceability Graph (RSTrace+)	Recommend reviewers on basis of traceability matrix	80-85%	[23]
5	Review Bot	Recommend reviewer based on experience & expertise	60-85%	[6], [7]
6	WhoDo	Suggest reviewer via load balancing approach	68%-72%	[3]

RQ6: What are the threats to the validity of the previous finding in the context of code reviewer recommendation tools?

Answer: We identified 14 tools in total, however, validity threats have been discussed for only 9 tools in their concerned studies. To elaborate on the validity threats, we may discuss the ‘CORRECT’ tool (Ser 1 of Table 9). One of its internal threats is that it is validated only for medium-sized projects and its external threat is that it has only experimented with projects that are developed on Python, Java and Ruby specifically. Similarly, ‘ADCR’ (Ser 2 of Table 9) internal threat is that its validation gives results on Gerrit repository projects specifically. While its external threat is in the context of the Cold start problem, i.e., a situation where newly joined users are difficult to recommend as they don’t have any detailed profile, experience, etc. So, external threat in the ‘ADCR’ tool is basically for newly joined code reviewers. The other external threat is the limitation of results of ADCR to 5 project datasets only. Moreover, further details are provided in Table 9.

3.4 Conclusion to Literature Review

Our SLR presents a systematic review of the literature concerning the domain of automated Code Reviewer Recommendation (CRR). 62 landmark studies (published from 2011 – March 2022) were investigated and analyzed after filtering them out via a stringent selection

criterion. For ease of analysis and to explore the studies as per the research questions, studies were further categorized into five groups i.e., CRR approaches, CRR Tool availability, CRR Validation Protocols, CRR Optimization Algorithms and a General category.

After thoroughly analyzing the selected research articles, 4 different CRR approaches were identified, 14 tools are found that are currently being used for automated CRR, 4 validation protocols are identified in the review and 25 optimization algorithms (single, multi and many-objective) have been alienated from the selected research articles. The aspects emphasized in this article welcome the researchers and practitioners to select the respective approach, tool, algorithm and validation protocol as per their research requirement/demand, as it may vary from case to case. However, based on our analysis, it may be safely concluded that the ‘Expertise and experience’ based CRR approach is widely used, whereas, ‘WhoReview’ is the best tool to automate reviewer recommendation due to higher accuracy (85-92%). Moreover, in terms of research productivity over the last 10 years, NSGA-II is the most frequently / widely used multi-objective optimization algorithm.

3.5 Summary Table of Literature Review

Table 11. Summary Table of Literature Review

S. No.	Ref. No.	Year	Author(s)	Tools/Framework	Algorithm	Number of objectives	Results
1.	[12]	2015	Xia, X., Lo, D., Wang, X., & Yang, X.	TIE	Text Mining and Similarity Model	1	Prediction Accuracy=85% MRR=64%
2.	[31]	2020	Sadman, N., Ahsan, M. M., & Mahmud, M. P.	ADCR	NLP Techniques	1	Training Accuracy=95% Validation Accuracy=94%
3.	[68]	2015	Jiang, J., He, J. H., & Chen, X. Y.	CoreDevRec	Support vector machines	1	Accuracy=80% MRR=0.63
4.	[16]	2018	Lipcak, J., & Rossi, B.	Source Code Reviewer Recommendation	Naïve Bayes	1	MRR= 59%
5.	[76]	2016	Rahman, M. M., Roy, C.	CORRECT	Code Reviewer	2	Precision=86%

			K., & Collins, J. A.		Ranking Algorithm		Recall=80%
6.	[74]	2019	Sülün, E., Tüzün, E., & Doğrusöz, U.	RSTrace+	Software Artifact Traceability Graphs	1	Recall=85% MRR=73%
7.	[16]	2015	Thongtanunam, P., Tantithamthorn, C., Kula, R. G., Yoshida, N., Iida, H., & Matsumoto, K. I	RevFinder	File path Similarity-Based	1	Accuracy=87% MRR=0.55
8.	[73]	2020	Strand, A., Gunnarson, M., Britto, R., & Usman, M.	Carrot	Machine learning-based Algorithm	1	Accuracy = 78%
9.	[60]	2013	Balachandran, V.	Review Bot	Statis Analyzer & Auto-reviewer Algorithm	1	Prediction Accuracy 59-92%
10.	[69]	2016	Ouni, A., Kula, R. G., & Inoue, K.	RevRec	GA	2	Precision=58% Recall=71% MMR=69%
11.	[72]	2019	Asthana, S., Kumar, R., Bhagwan, R., Bird, C., Bansal, C., Maddila, C., ... & Ashok, B.	WhoDo	Scoring Function + Load Balancer	2	Precision=51% Recall=64% F1-score=57%
12.	[9]	2020	Rebai, S., Amich, A., Molaei, S., Kessentini, M., & Kazman, R.	A Multi-Objective Code Reviewer Recommendation Framework	NSGA-II	3	Precision=60% Recall=53% MMR=68%
13.	[75]	2021	Chouchen, M., Ouni, A., Mkaouer, M. W., Kula, R.	WhoReview	Indicator-Based Evolutionary	2	Precision=68% Recall=77% MRR= 66%

			G., & Inoue, K.		Algorithm		
14.	[12]	2016	Bhutada, S., Balaram, V. V. S. S. S., & Bulusu, V. V.	Semantic Latent Dirichlet Allocation (SLDA)	Latent Dirichlet Allocation	1	F-measure=81% Purity=96%
15.	[2]	2015	Zanjani, M. B., Kagdi, H., & Bird, C.	cHRev	Reviewer - Expertise and File-Review Maps	1	Precision=59% Recall=48% F-Score=53%
16.	[16]	2019	Li, H. Y., Shi, S. T., Thung, F., Huo, X., Xu, B., Li, M., & Lo, D.	Deep Review	Neural Network-Deep Multi-instance Learning	1	F1 score=0.49 Area Under the Curve=0.87
17.	[77]	2016	Almhana, R., Mkaouer, W., Kessentini, M., & Ouni, A.	Recommending Relevant Classes for Bug Reports using Multi-objective Search	NSGA-II	2	Precision=89% Recall=72% Accuracy=68%
18.	[31]	2017	Rahman, M. M., Roy, C. K., & Kula, R. G.	RevHelper	Random Forest	1	Precision=74% Recall=78% F1-Score=63% Accuracy=65%

Table 11, presents a summary of the literature on different tools/frameworks that are proposed by worthy researchers. It also provides crisp and to-the-point knowledge about CRR tools/frameworks as the Algorithm used, the number of objectives defined, results, author's information, publication year and cited reference number.

3.6 Research gap

This section discusses the research gaps and limitations encountered in previous literature. Analysis was done on around 62 selected primary studies (in terms of tools/frameworks, algorithms, validation protocols, etc.), after an extensive screening process to look for research that provides a recommendation for code reviewers for code review.

The gap found in our selected studies was that no research focused on using a multi-objective algorithm i.e., NSGA-III (an extended form of NSGA-II, proposed in early 2014) to recommend code reviewers by taking 3 objectives into account simultaneously in the code reviewing process. There was no single fully automated tool/framework proposed to facilitate reviewers to do this cumbersome job. Though partially proposed or semiautomated tools i.e., models and algorithm's pseudocode available in literature but no framework is proposed using them.

We found only one study [9], where authors worked on 3 objectives to optimize them using NSGA-II. But on contrary, many researchers have found that NSGA-II does not perform well on problems with more than two objectives. This is why [65], [82], [44] and [45] have then worked on NSGA-III to provide a method for problems with more than two objectives. We are adding our effort to that research gap by discussing and validating the potential of NSGA-III to optimize 3 fitness functions simultaneously using NSGA-III. Also, to estimate and compare the execution time of both the algorithm for the same experiments. The goal is to use the same 3 objectives as used by [9] and then provide a comparative analysis with this research and other state-of-the-art as well to find a better solution to this gap.

PROPOSED APPROACH

Chapter 4 : PROPOSED APPROACH

In this section, the recommendation of a most appropriate set of reviewers is presented for pull requests to be reviewed as a framework/approach by using the NSGA-III optimization algorithm. The proposed approach consists of data collection, data pre-processing, main components of the approach, detailed knowledge about NSGA-III (i.e., high-level pseudocode) algorithm, solution representation, fitness functions and change operators.

The recommendation of reviewers is performed by using the NSGA-III optimization algorithm. For validation and visualization in form of graphs, NSGA-II is implemented as well in comparison to compare the results of both. Figure 9 shows the fundamental steps of the proposed approach.

The methodology starts with a project's new pull request to be reviewed is received. The data is collected from the project in terms of its previous pull request information (newly opened, under-review, closed), reviewers' and developers' information, etc.

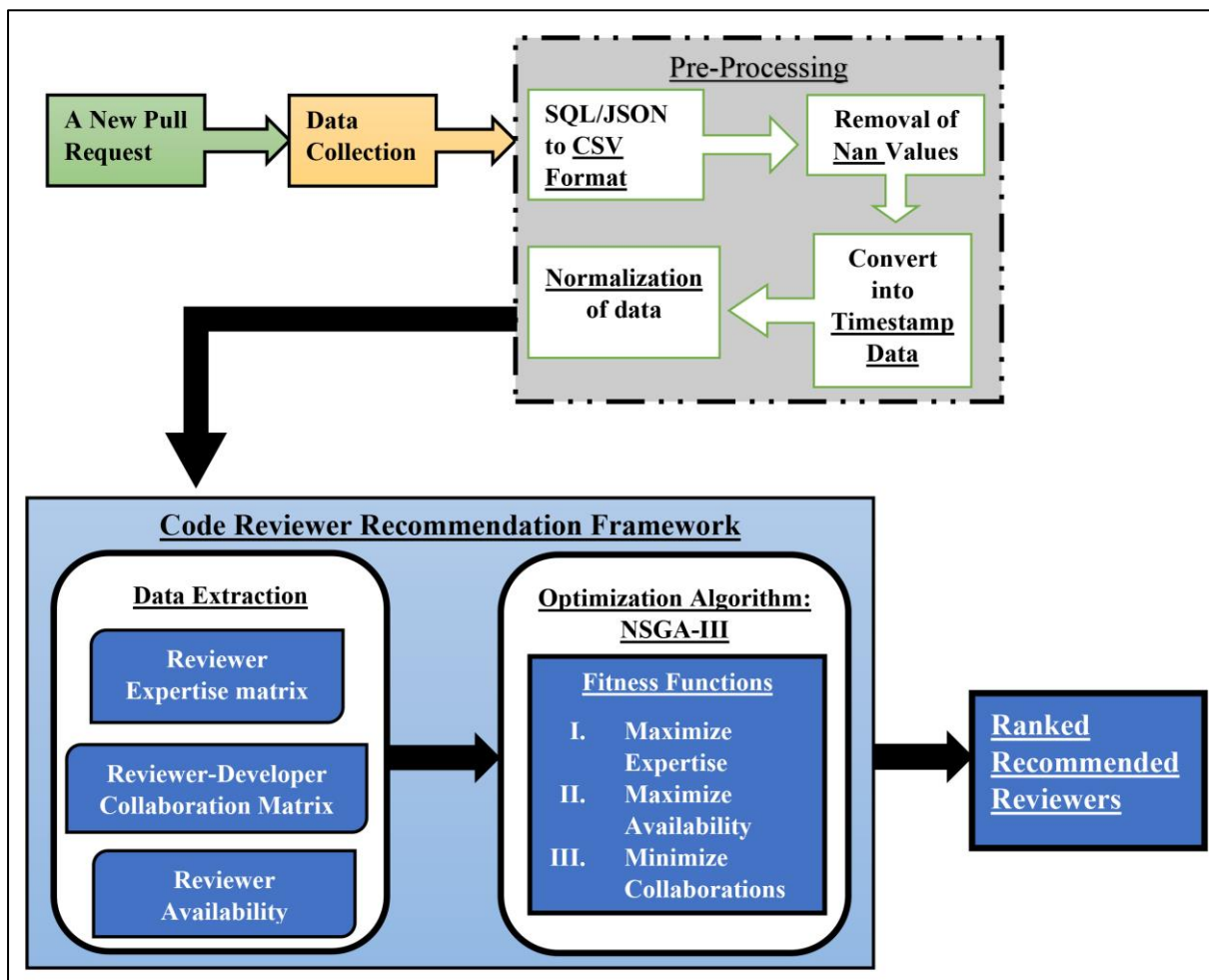


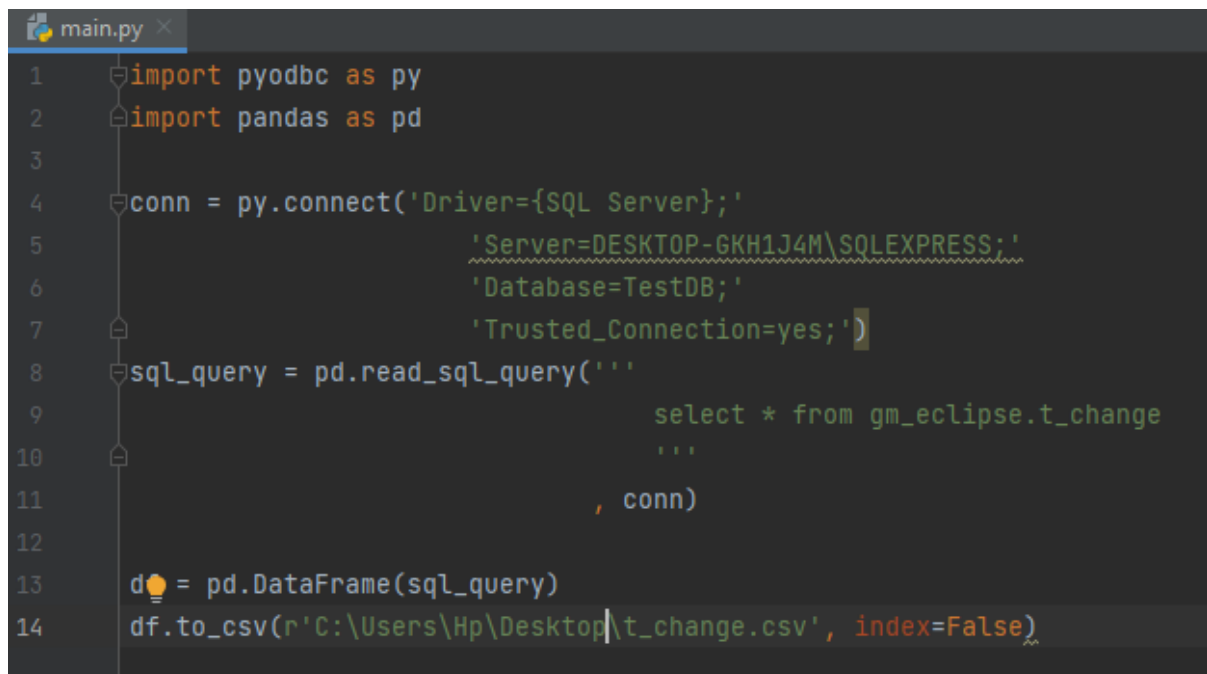
Figure 9. Overview of multi-objective search-based approach for CRR

4.1 Data Pre-Processing

To clean data and organize it according to the requirement of the framework, pre-processing is performed.

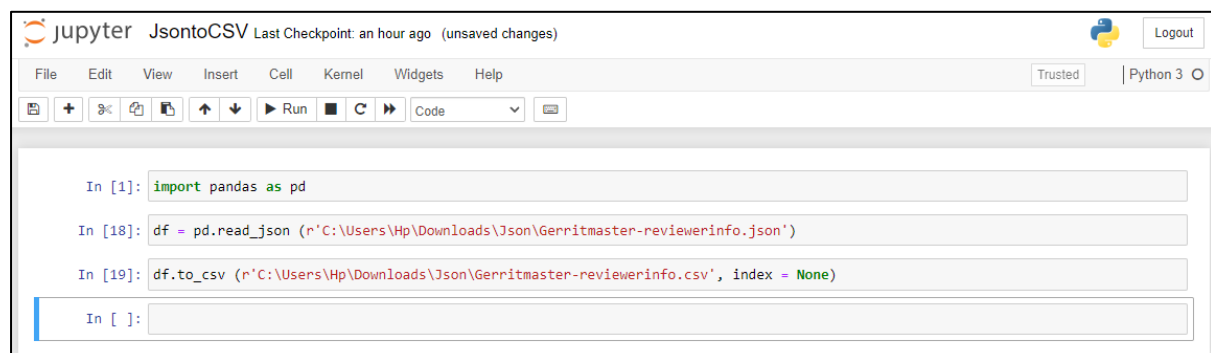
4.1.1 Conversion of SQL & JSON to .CSV Format

The first step of data preprocessing is to convert the data into .CSV file format as it is faster to handle, standard format, and easy to parse. We find some datasets that are in SQL format and sometimes we use APIs to fetch data from ongoing live projects. Thus, the conversion of data is very important in a standard format to use in the proposed framework. Figure 10 and 11 shows the code snippet by which SQL and JSON are converted into .CSV file format.



```
main.py x
1 import pyodbc as py
2 import pandas as pd
3
4 conn = py.connect('Driver={SQL Server};'
5                  'Server=DESKTOP-GKH1J4M\SQLEXPRESS;'
6                  'Database=TestDB;'
7                  'Trusted_Connection=yes;')
8 sql_query = pd.read_sql_query(''
9                               'select * from gm_eclipse.t_change
10                              ...
11                              , conn)
12
13 df = pd.DataFrame(sql_query)
14 df.to_csv(r'C:\Users\Hp\Desktop\t_change.csv', index=False)
```

Figure 10. SQL to .CSV Format



```
Jupyter JsonoCSV Last Checkpoint: an hour ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [1]: import pandas as pd
In [18]: df = pd.read_json(r'C:\Users\Hp\Downloads\Json\Gerritmaster-reviewerinfo.json')
In [19]: df.to_csv(r'C:\Users\Hp\Downloads\Json\Gerritmaster-reviewerinfo.csv', index = None)
In [ ]:
```

Figure 11. JSON to .CSV Format

4.1.2 Removal of NaN Values

NaN is known as the Not-a-Number value. It represents the missing/absence of value in a cell. It is a type of special floating-point value that cannot be converted into any other type than float. In our framework, NaN values are replaced by zero. The difference between zero and NaN is that zero is a value but NaN represents the absence of any value. The **purpose** of removing NaN values is because the optimization algorithm does not perform well or the accuracy of the algorithm can't be made sure on NaN values data. Figure 12 shows the data with NaN values (left side) and data when replaced with zero (right side). It may also be noticed that we found NaN values only in the 'Closed' date and time of pull request data.

Closed	Closed
NaN	0
NaN	0
NaN	0
NaN	0
NaN	0
NaN	...
...	2015-06-02 15:42
2015-06-02 15:42	2014-11-24 15:08
2014-11-24 15:08	2014-11-11 14:03
2014-11-11 14:03	2015-01-07 11:49
	2014-10-23 14:01

Figure 12. NaN Value (left) and Removed NaN Values (right)

4.1.3 Timestamp Data

The DateTime data is converted into timestamp data in preprocessing. This is an optional step (if required according to the dataset) to perform while pre-processing data. The purpose of the step is to follow the required standard format of the data frame i.e., timestamp otherwise it shows an error to convert it into a timestamp. Figure 13 shows the data in Date-Time format (left) and Timestamp formatted data (right).

Created	Closed	Created	Closed
2022-01-11 12:28	0	2022-01-11 12:28:00	1970-01-01 00:00:00
2021-12-30 13:51	0	2021-12-30 13:51:00	1970-01-01 00:00:00
2021-07-01 10:30	0	2021-07-01 10:30:00	1970-01-01 00:00:00
2020-06-18 05:26	0	2020-06-18 05:26:00	1970-01-01 00:00:00
2020-06-18 05:18	0	2020-06-18 05:18:00	1970-01-01 00:00:00
...
2014-02-03 17:00	2015-06-02 15:42	2014-02-03 17:00:00	2015-06-02 15:42:00
2014-02-03 16:20	2014-11-24 15:08	2014-02-03 16:20:00	2014-11-24 15:08:00
2014-02-03 13:58	2014-11-11 14:03	2014-02-03 13:58:00	2014-11-11 14:03:00
2014-01-15 23:01	2015-01-07 11:49	2014-01-15 23:01:00	2015-01-07 11:49:00
2014-01-14 14:08	2014-10-23 14:01	2014-01-14 14:08:00	2014-10-23 14:01:00

Figure 13. Date-Time Data (left) and Timestamp formatted data (right)

4.1.4 Normalization

In the last step of pre-processing, Data was normalized by doing multiple steps.

- a) Some unwanted columns were dropped from the data that were of no use according to the framework. By doing this, the computations got quicker and the load of the dataset was reduced.
- b) Normalization of two columns ‘Status’ and ‘Priority’ has been done as they have string values (high, low, medium) repetitively in the data. So, to faster the computations we marked them with integers as (0, 1, 2). Table 12 shows both columns and their normalized values.
- c) Lastly, the data (fitness values) for displaying the results are normalized between the values ranging from 0 to 1. This type of normalization is called Min-Max scaling. In our proposed framework, the NumPy Python library is used to implement its scaling function. The purpose of Min-Max scaling is to remove any outlier from the dataset.

After performing normalization, fitness values would not be less than 0 and the maximum value wouldn't be greater than 1. Figure 14 displays the (left) un-normalized data where the values of fitness are very less. On the right side, the fitness values are normalized between zero to one.

Table 12. Data Normalization

Status		Priority	
Value in Dataset	Value Normalized	Value in Dataset	Value Normalized
In progress	0	Urgent	0
Feedback	1	High	1
Rejected	2	Intermediate	2
Resolved	3	Normal	3
Closed	4	Low	4

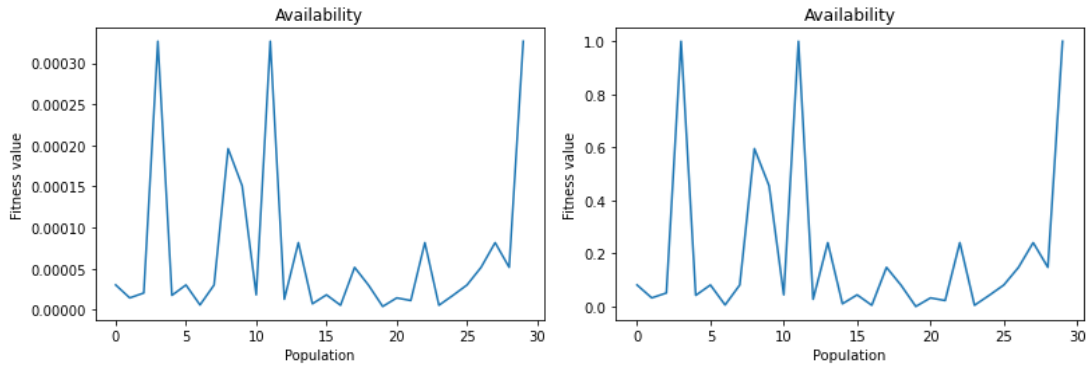


Figure 14. Un-normalized data (left) and Normalized data (right)

4.2 Code Reviewer Recommendation (CRR) Framework

This section presents the framework in two steps. Step 1 explains the data extraction or major components of the approach while step 2 explains the optimization algorithm i.e., NSGA-III and fitness functions, etc.

4.2.1 Major Components of the Approach

There are 3 major components of the approach as shown in the methodology figure. The purpose is to extract the data from the dataset by making the respective matrix as required by the framework. Collectively there are major three components of our approach that are defined below:

4.2.1.1 Reviewer Expertise Matrix

This component helps the framework in identifying the reviewer and file connections. This matrix can represent the expertise of a reviewer with the help of previous commits and those pull requests that have been closed. Expertise is defined as the number of times a reviewer reviewed/is familiar with the same file. Matrix keeps the track record of every reviewer that worked on a specific file and the number of times reviewed that particular time.

FR (File-Reviewer) is a $P \times M$ matrix where each entry is stored as $fr_{(k,i)}$ (number of times reviewer r_i reviewed the file f_k . P is the total number of files requested to be reviewed and M is the total number of reviewers working on a project, where $i \in \{1, 2, \dots, M\}$ and $k \in \{1, 2, \dots, P\}$.

$$FR = (fr_{(k,i)}) \in P \times M \quad \text{Equation 1 [9]}$$

4.2.1.2 Reviewer Developer Collaboration Matrix

The collaboration between a reviewer and developer is presented in this component of the approach. The history of closed pull requests is used to present this matrix. For every reviewer, the number of times they collaborated with the developers together and the number of files reviewed are extracted. The collaboration matrix (*DR*) is defined as $N \times M$ where each entry $dr_{(j,i)}$ represents the number of times reviewer r_i reviewed a file, changed by a developer d_j where $i \in \{1, 2, \dots, M\}$ and $j \in \{1, 2, \dots, N\}$, M is the total number of reviewers and N is the total number of developers.

$$DR = (dr_{(j,i)}) \in N \times M \quad \text{Equation 2 [9]}$$

4.2.1.3 Reviewer Availability

The availability of a reviewer is defined as the workload/number of files per open pull request where they are currently involved. Availability is a vector $A = [a_1, a_2, \dots, a_m]$ where a_i represents the total number of files of the open pull of a reviewer.

4.2.2 Multi-Objective Optimization (NSGA-III)

Multi-objective search is defined as ‘the optimization of more than one objective simultaneously’. But it is difficult to find a solution that results in an optimal one as the objectives are conflicting. As result, a multi-objective search algorithm is a solution that could provide a set of different solutions. Since in our framework we are proposing multi objectives to be optimized and provide more than one solution, non-dominated sorting genetic algorithm-III (NSGA-III) is an algorithm that has demonstrated its usefulness in three or more objectives.

The pseudocode of NSGA-III is explained in Algorithm 1 [83].

NSGA-III starts with an input, randomly initialized population with size N , a set of H well-distributed reference points that are chosen on a unit hyperplane using Das and Dennis's method [83]. At a generation S_t , complete population P_t produces new offspring population Q_t with the help of mutations and recombination operators in which everyone population member is associated with each reference point and any selection operator will allow competition to be set among different reference points. A combined population $R_t = P_t \cup Q_t$ is then formed. So, we have got the first non-dominated solution P_{t+1} until every solution cannot be included from the whole front. All members of the merged population, at every generation, are normalized using a systematic extreme point update strategy mainly by using population-minimum and population-maximum objective values. Each member is then associated with a particular reference direction using the orthogonal distance of a member to a reference direction. Thereafter, a niching methodology is used to choose a diverse set of solutions by providing equal emphasis to each reference direction. The algorithm is repeated until the stopping criteria are matched.

Key Points:

- ❖ NSGA-III performs selection if and only if at least one of the two individuals being compared is infeasible. In that case, NSGA-III prefers feasible over infeasible and less violating over more violating individuals.
- ❖ To maintain better Coverage of solutions NSGA-III uses a reference point mechanism while NSGA-II uses crowding distance calculation.
- ❖ NSGA-III (Deb & Jain, 2014; Jain & Deb, 2014) uses to pre-allocated reference set mechanism to choose better diverse solutions in the size of the population in free space, whereas the NSGA-II algorithm does not require any pre-allocated methods on the objective space. So, with more time taken to generate the first solution in spaces, NSGA-III has easily generated the first solution so NSGA-III is better than the NSGA-II algorithm.

Algorithm 1: High-level Pseudocode for NSGA-III [83]
Input: H structured reference point Z^s and parent population P_t (total population)
Output: P_{t+1} (non-dominated solution)/Population P
1: $S_t = \emptyset, i=1$
2: $Q_t = \text{Recombination} + \text{Mutation} (P_t)$

```

3:  $R_t = P_t \cup Q_t$ 
4:  $(F_1, F_2, \dots) = \text{Non-dominant-sort}(R_t)$ 
5: repeat
6:  $S_t = S_t \cup F_i$  and  $i = i + 1$ 
7: until  $|S_t| \geq N$ 
8: The last front to be included:  $F_l = F_i$ 
9: if  $|S_t| = N$  then
10:  $P_{t+1} = S_t$ , break
11: else
12:  $P_{t+1} = U_{j=1}^{l-1} F_j$ 
13: Points to be chosen from  $F_l$ :  $K = N - |P_{t+1}|$ 
14: Normalize fitness objectives and create a reference set  $Z^l$ : Normalize  $(f^n, S_t, Z^s)$ 
15: Associate each member  $s$  of  $S_t$  with a reference point [closed reference point, the distance between  $s$  and closed reference point]
16: Compute niche count of reference point and then  $Q_{t+1} = \text{create new pop}(P_{t+1})$ 
17:  $t=t+1$ 
18: end if

```

4.2.3 Fitness Functions

Since we are working on a multi-objective optimization so in our approach, we intend to provide optimization on three fitness functions. These three objective functions are Expertise, Availability, and Collaboration. We aim to maximize the formulation of expertise and availability of the reviewers while minimizing the fitness function of collaboration due to socio-technical aspects of reviewers and developers in the hope to reduce human biases.

The approach in this thesis presents the output of a set of non-dominated solutions. The **motivation** behind using multi-objective functions is a recent observation that was studied at Microsoft [51], that highlights promoting a variety of norms of the team. For example; some teams prefer to be diverse and some prefer to have close connections. Sometimes the better connection leads to a more valuable result in less time and on some occasions, it leads to human biases or quick code approval due to shorter deadlines (if allied with low expertise).

It is up to the team member to make a change in collaboration function if needed according to their preferences. The goal is to provide a varied range of good reviewers as output rather

than one solution so that a user could select the reviewer according to his/her needs. The details of fitness functions are explained in the below sub-sections:

4.2.3.1 Availability

Availability is defined as the inverse of approximated wait until reviewers from their workload (already working on a set of file S) become available. In our approach, the workload is considered as the collection of the number of commits submitted lately within the last seven days and the total amount of all open pull requests.

$$\text{Availability} = \frac{1}{\sum_{k=1}^P \sum_{i=1}^M a_i * S[k,i]}, s_{k,i} > 0 \quad \text{Equation 3 [9]}$$

In the equation, $a = \{a_1, a_2, \dots, a_m\}$ for contains the task that are in queue for a reviewer. The tasks queued for a reviewer r_i are represented by a_i .

P= total number of files requested to be reviewed

M= total number of reviewers

4.2.3.2 Expertise

File urgency is an important column in the dataset that defines the status of priority a feel needs to be reviewed. The file would take a score of urgency/priority from the pull request as defined in Table no 12. FR (File reviewer) and PR (File priority) would both be used to define the expertise function.

$$\text{Expertise} = \sum_{k=1}^P \sum_{i=1}^M \frac{FR[k,i] + PR[k]}{S[k,i]}, s_{k,i} > 0 \quad \text{Equation 4 [9]}$$

Where M= total number of reviewers

P= total number of developers

FR= File-Reviewer matrix

S [k, i] = rank of the reviewers

Note: A reviewer having a rank of 2 would be more appropriate/expert for reviewing in comparison to a reviewer having a rank of 6 or 7. The lesser the rank, the more appropriate the reviewer, and the more the availability and expertise both, as the rank is in the denominator of the formula in both functions. Rank is inversely proportional to the expertise and availability of the reviewer. Reviewers with maximum expertise and availability would have more chances to sustain in the next generation of the algorithm to produce more and better results. And the final output of set of peer reviewers would definitely be the reviewers having best expertise.

4.2.3.3 Collaboration

This is the only objective which we aim to minimize, for the factor of biasness to avoid. It is the summation of all associates between the recommended reviewers chosen to work with a selected set of developers.

$$\mathbf{Collaboration} = \sum_{k=1}^N \sum_{j=1}^P \sum_{i=1}^M \mathbf{DR}[j, i] * \mathbf{FD}[k, j] * (\mathbf{S}[k, j] > 0) \quad \mathbf{Equation 5 [9]}$$

In the equation above,

P=total number of files requested to be reviewed

M=total number of reviewers

N=total number of developers

S [k, j] = rank of reviewer

DR=Developer-Reviewer Matrix

FD=File-Developer Matrix

Both the matrix FD and DR are extracted during the data extraction step as defined in the approach diagram.

IMPLEMENTATION, RESULTS & DISCUSSION

Chapter 5 : IMPLEMENTATION, RESULTS & DISCUSSION

This section provides details about the implementation, results of the different projects, and their discussion. The implementation consists of data gathering of different ongoing opensource projects for validation and results formulation. It also consists of basic information relevant to implementation/experimental setup i.e. (language, tool, projects, and framework used). The framework has been analyzed and discussed in this section.

The results were compared to the state-of-the-art where the worthy researchers have used the same datasets in mono, bi and triple objectives with other algorithms. We only reimplemented NSGA-II to compare the results with our NSGA-III triple objectives, as authors [9] didn't provide their code, fitness values, population and generation values and graphical representations to compare.

5.1 Data Collection/Dataset

We evaluated our approach on 3 different ongoing open-source projects as LibreOffice, Qt, and OpenStack. Two datasets (LibreOffice and Qt) were last updated till 2016 and one was last updated till 2018 in SQL format only. To bring versatility and add novelty to datasets, we collected the updated data about the closed pull request till December 2021. Table 13 show the summary for open-source validation projects that includes number of reviewers, number of pull requests (closed and newly opened), duration and repository link. LibreOffice is an opensource software used for formal documentation (word, spreadsheets, slides and diagrams). Qt is a multi-platform software used for creating different user interfaces. Lastly, OpenStack is a cloud computing-based software used for virtual servers and other resources to made available for end-users. We selected these 3 projects as they have been a part of literature frequently [9], [29], [11], [51], [69] and our comparison would be a strong result. Also, these projects contain large number of code reviews.

Pre-Cap of Data Collection: For expertise and collaborations fitness function explained in methodology, we considered all the data since the start of the project because we accept as truth that more information about the expertise and collaborations of the developers is useful in assigning the appropriate reviewer set in the end.

Regarding the reviewer availability and to approximate the current workload of them, we considered the last 7 days of open pull requests.

The collection of data was gathered by three different ways.

The complete Dataset shared in .CSV format is now shared and available in dataset.rar file https://drive.google.com/file/d/1CiZWHc0z_JFZ_Tg7pewTfNZb91o9XzIg/view?usp=sharing

5.1.1 SQL Format

First the SQL format files that were shared by previous authors [9], [11], [29], [51], [69] were downloaded from the link <https://kin-y.github.io/miningReviewRepo/>. The SQL files were converted into .CSV as explained in Data preprocessing step and then used.

5.1.2 CSV Format

The closed pull request after 2016 and 2018 were downloaded from the accounts of projects by registering with them as reviewer/developer. The .csv format files were accessed or download (closed pull request only). The downloaded files were merged with the CSV's that were gathered after conversion from SQL files (Section 5.1.1).

5.1.3 Postman API Platform

The newly opened pull request was not available in .CSV format on accounts of projects. We collected data in JSON format of past 7 days using an HTTP request of Postman API Platform via an HTTP request as shown in Example pull request (Figure 15). The JSON files were converted into .CSV as well in part of preprocessing of data.

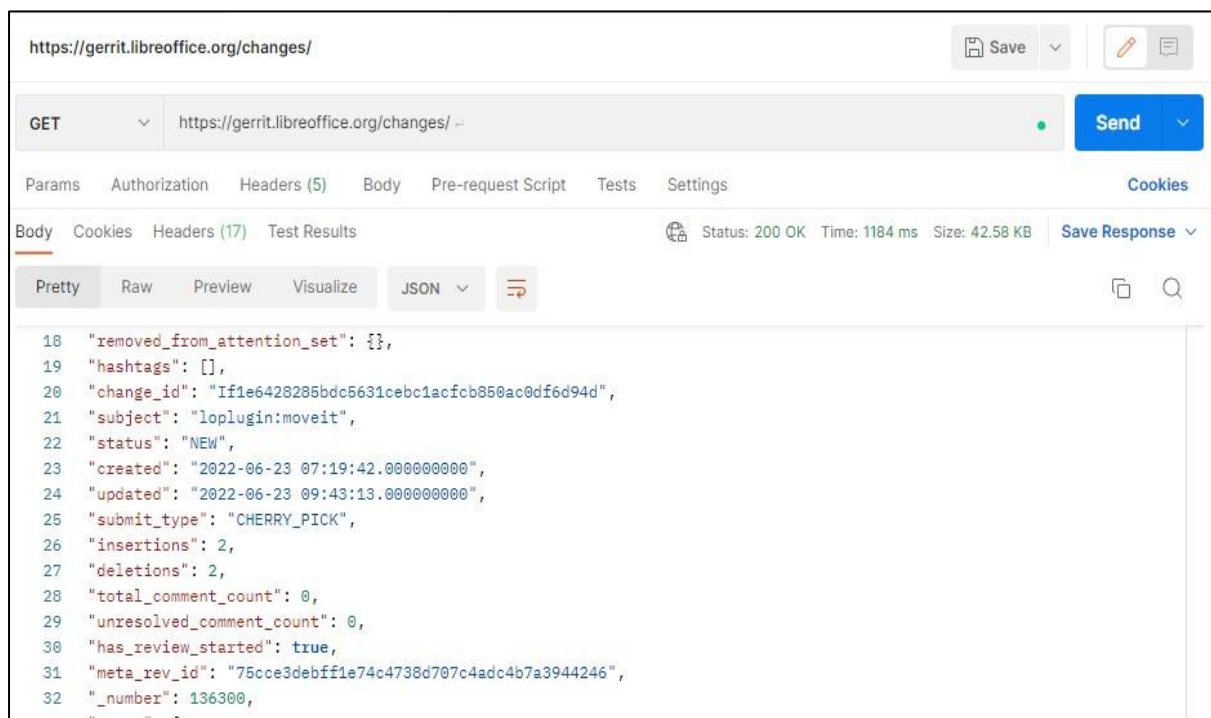


Figure 15. An example of Postman Data Collection

Table 13. Summary Table of Open-Source Validation Projects

Project name	Repository link	Project Duration	Num of pull request (Closed)	Num of pull request (New-Past 7 days)	Num of reviews
LibreOffice	https://git.libreoffice.org/core	07/2014~ 12/2021	28030	12	934
Qt	https://code.qt.io/cgit/qt/qtbase.git/	05/2011~ 11/2021	115888	25	1437
OpenStack	https://opendev.org/openstack	07/2011~ 12/2021	173749	52	5091

5.2 Experimental Setup and Research Questions

The proposed approach is implemented in Python Language on Jupiter Lab (3.0) using Pymoo library [84] for algorithm working and visualization. The overview of algorithm is provided in methodology (Section 4.2.2). While defining problem and Pymoo algorithm library, the change or genetic operators are used. We applied Das-Dennis [83] approach to define reference points. The rest of the genetic operator used in results are explained in table 14. The literature results are computed using random selection (to compare with state of the art), we have used tournament selection in one experiment (to give a new horizon for future comparison). Total 7 experimental sets are made in which first 5 are with random selection, other two with tournament selection. Experiment 7 is a try and check set for abnormal behavior of algorithms.

We used ‘Uniform (‘real_ux’, ‘bin_ux’, ‘int_ux’) Crossover’ and ‘Polynomial (‘real_pm’, ‘int_pm’) Mutation’ to explore and exploit the search space. Mutation prevents all solutions in the population falling into a local optimum. Also, it helped to make us sure with the results as search algorithms are stochastic/probabilistic in nature.

The final output of the algorithm is a set of solutions representing trade-offs between the three objectives. It is up to the manager to select the reviewer’s assignment (choose a solution) based on their preferences. In general, the preferences are defined based on the current context: urgency to release code quickly, available resources, speedy growth phase of

the project, etc. These different contexts are not changing daily and they are not related to only one or few pull-requests.

Table 14. Experimental Genetic Operators

Experiment Number	Pop Size	Selection Operator	Crossover Operator	Crossover Probability	Mutation Operator	Mutation rate	Number of Generation
Exp. 1	50	Random	Uniform	0.5	Polynomial	0.01	100
Exp. 2	50	Random	Uniform	0.6	Swap	0.1	80
Exp. 3	100	Random	Uniform	0.8	Polynomial	0.2	100
Exp. 4	200	Random	Uniform	0.5	Polynomial	0.1	230
Exp. 5	150	Random	Uniform	0.8	Swap	0.05	200
Exp. 6	80	Tournament (Selection pressure = 2)	Uniform	0.85	Polynomial	0.5	120
Exp. 7	100	Random / Tournament	Uniform	0.5	Polynomial	0.5	50

5.2.1 Research Questions

The research questions are presented for this thesis to represent results in a classical way to estimate the effectiveness of our approach.

RQ1- Does the proposed approach efficiently and precisely identify the relevant code reviewers?

RQ2- Compare the proposed approach performance, if it is significantly better than the existing tools or frameworks (mono-objective approach of all objectives, search algorithms and other variants of GA)? Also compare the execution time for finding best solutions by NSGA-II and NSGA-III on the same experiments?

To answer research question 1, the validation is performed on 3 medium sizes to large size open source project to evaluate the efficiency of our CRR approach. To follow the fair comparison with previous work, we took recently closed reviews and their reviewers assigned to these pull request. Every fitness function was run by setting them together in a defined problem. The efficiency is validated in result of Precision@Exp., Recall@Exp., MRR and Average Accuracy.

We calculate the precision and recall as follows.

$$\mathbf{Precision@Exp} = \frac{TP}{TP+FP} \quad \mathbf{Equation 6 [11]}$$

$$\mathbf{Recall@Exp} = \frac{TP}{TP+FN} \quad \mathbf{Equation 7 [11]}$$

where TP (True Positive) corresponds to the number of top-k reviewers recommended by the approach and also actual reviewers;

FP (False Positive) corresponds to the number of top-k reviewers recommended by the approach, but not actual reviewers;

FN (False Negative) corresponds to the number of actual reviewers, that are not among the top-k reviewers recommended by the approach.

TN (True Negative) corresponds to the number of not actual reviewers, that are also not among the top-k reviewers recommended by the approach.

MRR (Mean Reciprocal Rank): The average rank of correct reviewers in the recommendation list. The higher the MRR, the better the rank recommendation. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of recommendation list.

Given a reviewers recommendation lists R, the score MRR is calculated as follows:

$$\mathbf{MRR} = \frac{|\sum_{r \in R} \text{rank}(r)|}{|R|} \quad \mathbf{Equation 8 [9]}$$

where rank(r) is the rank score of the first reviewer in the recommendation list r. The higher is the MRR score, the better is the recommendation approach.

Average Accuracy of Projects: The projects accuracy is determined as average accuracy in terms of number of experiments performed in that project. The average accuracy is determined as:

$$\mathbf{Average Accuracy} = \frac{\sum \text{Accuracy}(E^1 + E^2 \dots E^N)}{N} \quad \mathbf{Equation 9 [51]}$$

Where E^1 is the accuracy of experiment 1, E^2 is the accuracy of experiment 2 till last Experiment. The N is the total number of Experiments used in a project.

The Accuracy of each experiment is calculated as:

$$\mathbf{Accuracy} = \frac{TP+TN}{TP+TN+FN+FP} \quad \mathbf{Equation 10 [51]}$$

To answer research question 2, we compared our above metrics with the state-of-the-art. We compare our results with WhoReview (IBEA Algorithm), REVFINDER, ReviewBot and some other variants of GA were compared as well (GA, NSGA-II AEC). The Comparison of our approach with NSGA-II was done using execution time calculation and graphical visualization for showing maximize, minimize and premature convergence rate of fitness function with different genetic operators.

We also calculated the Mean fitness for all the experiments for NSGA-III and NSGA-II. For expertise and availability fitness functions: the greater the value (nearer to 1), better the fitness. And for collaboration (since we are minimizing it), the lesser the value, the better the fitness. Also, we measured and compared the execution times for both NSGA-II and NSGA-III to compute the comparison of time factor as well.

$$\text{Mean Fitness} = \frac{\Sigma \text{Fitness_values}}{\text{Popsize}} \quad \text{Equation 11 [69]}$$

Where Σ is the sum of all fitness values against every pop size divided by total pop-size.

Confusion matrix for LibreOffice Experiment no 6:

LibreOffice Confusion Matrix		Actual Values		
		Recommended	Not Recommended	Total
Predicted Values	Recommended	TP=210	FP=99	309
	Not Recommended	FN=68	TN=123	191
	Total	278	222	500

Precision Calculation: $TP/(TP+FP) \rightarrow 210/(210+99) \rightarrow 0.679 \rightarrow 67\%$

Recall Calculation: $TP/(TP+FN) \rightarrow 210/(210+68) \rightarrow 0.755 \rightarrow 75\%$

Accuracy Calculation: $TP+TN/(TP+TN+FP+FN) \rightarrow 210+123/(210+123+99+68) \rightarrow 0.66 \rightarrow 66\%$

Confusion matrix for Qt Experiment no 6:

Qt Confusion Matrix		Actual Values		
		Recommended	Not Recommended	Total
Predicted Values	Recommended	TP=194	FP=87	281
	Not Recommended	FN=93	TN=126	219
	Total	287	213	500

Precision Calculation: $TP/(TP+FP) \rightarrow 194/(194+87) \rightarrow 0.690 \rightarrow 69\%$

Recall Calculation: $TP/(TP+FN) \rightarrow 194/(194+90) \rightarrow 0.675 \rightarrow 67\%$

Accuracy Calculation: $TP+TN/(TP+TN+FP+FN) \rightarrow 194+126/(197+126+87+93) \rightarrow 0.64 \rightarrow 64\%$

Confusion matrix for OpenStack Experiment no 6:

OpenStack Confusion Matrix		Actual Values		
		Recommended	Not Recommended	Total

Predicted Values	Recommended	TP=246	FP=61	307
	Not Recommended	FN=37	TN=156	193
	Total	283	217	500

Precision Calculation: $TP/(TP+FP) \rightarrow 246/(246+61) \rightarrow 0.803 \rightarrow 80\%$

Recall Calculation: $TP/(TP+FN) \rightarrow 246/(246+37) \rightarrow 0.869 \rightarrow 86\%$

Accuracy Calculation: $TP+TN/(TP+TN+FP+FN) \rightarrow 246+156/(246+156+61+37) \rightarrow 0.804 \rightarrow 80\%$

Similarly, all other confusion matrix for other experiments is made and precision, recall and accuracy values are calculated as shown in Table 15 and 16. The highlighted/bold values in both the tables are the best precision and recall values found in all three projects.

Table 15. Precision Results

Project Name	Experiment number	Precision@Exp.					
		Proposed Approach (NSGA-III)	AEC (NSGA-II) [9]	RevRec (GA) [69]	Who Review (IBEA) [51]	RevFinder [11]	ReviewBot [29]
LibreOffice	Exp. 1	0.64	N/A	0.52	0.61	0.48	0.38
	Exp. 2	0.59	N/A	0.45	0.54	0.4	0.36
	Exp. 3	0.57	N/A	0.50	0.56	0.42	0.40
	Exp. 4	0.52	N/A	0.41	0.53	0.32	0.32
	Exp. 5	0.51	N/A	0.39	0.46	0.3	0.23
	Exp. 6	0.67	N/A	N/A	N/A	N/A	N/A
Qt	Exp. 1	0.66	0.58	0.49	0.58	0.3	0.22
	Exp. 2	0.62	0.51	0.42	0.53	0.27	0.19
	Exp. 3	0.57	0.54	0.45	0.55	0.29	0.13
	Exp. 4	0.55	0.52	0.41	0.43	0.21	0.10
	Exp. 5	0.50	0.46	0.34	0.48	0.16	0.09
	Exp. 6	0.69	N/A	N/A	N/A	N/A	N/A
OpenStack	Exp. 1	0.74	0.70	0.59	0.62	0.32	0.24
	Exp. 2	0.69	0.64	0.57	0.55	0.27	0.2
	Exp. 3	0.67	0.65	0.51	0.59	0.30	0.22
	Exp. 4	0.63	0.63	0.43	0.54	0.25	0.16
	Exp. 5	0.57	0.54	0.46	0.48	0.21	0.11

	Exp. 6	0.80	N/A	N/A	N/A	N/A	N/A
--	---------------	-------------	-----	-----	-----	-----	-----

Table 16. Recall@Exp Results

Project Name	Experiment number	Recall@Exp.					
		Proposed Approach (NSGA-III)	AEC (NSGA-II) [9]	RevRec (GA) [69]	Who Review (IBEA) [51]	RevFinder [11]	ReviewBot [29]
LibreOffice	Exp. 1	0.67	N/A	0.34	0.48	0.32	0.18
	Exp. 2	0.62	N/A	0.48	0.52	0.38	0.22
	Exp. 3	0.59	N/A	0.57	0.56	0.42	0.20
	Exp. 4	0.59	N/A	0.58	0.61	0.45	0.31
	Exp. 5	0.69	N/A	0.59	0.68	0.49	0.38
	Exp. 6	0.75	N/A	N/A	N/A	N/A	N/A
Qt	Exp. 1	0.58	0.56	0.41	0.44	0.14	0.09
	Exp. 2	0.62	0.60	0.50	0.45	0.27	0.16
	Exp. 3	0.68	0.66	0.55	0.58	0.30	0.20
	Exp. 4	0.72	0.68	0.59	0.60	0.35	0.24
	Exp. 5	0.73	0.70	0.65	0.64	0.43	0.30
	Exp. 6	0.67	N/A	N/A	N/A	N/A	N/A
OpenStack	Exp. 1	0.69	0.59	0.31	0.46	0.15	0.12
	Exp. 2	0.70	0.68	0.39	0.49	0.29	0.2
	Exp. 3	0.78	0.76	0.52	0.59	0.37	0.32
	Exp. 4	0.80	0.75	0.54	0.60	0.46	0.39
	Exp. 5	0.82	0.80	0.66	0.63	0.50	0.41
	Exp. 6	0.86	N/A	N/A	N/A	N/A	N/A

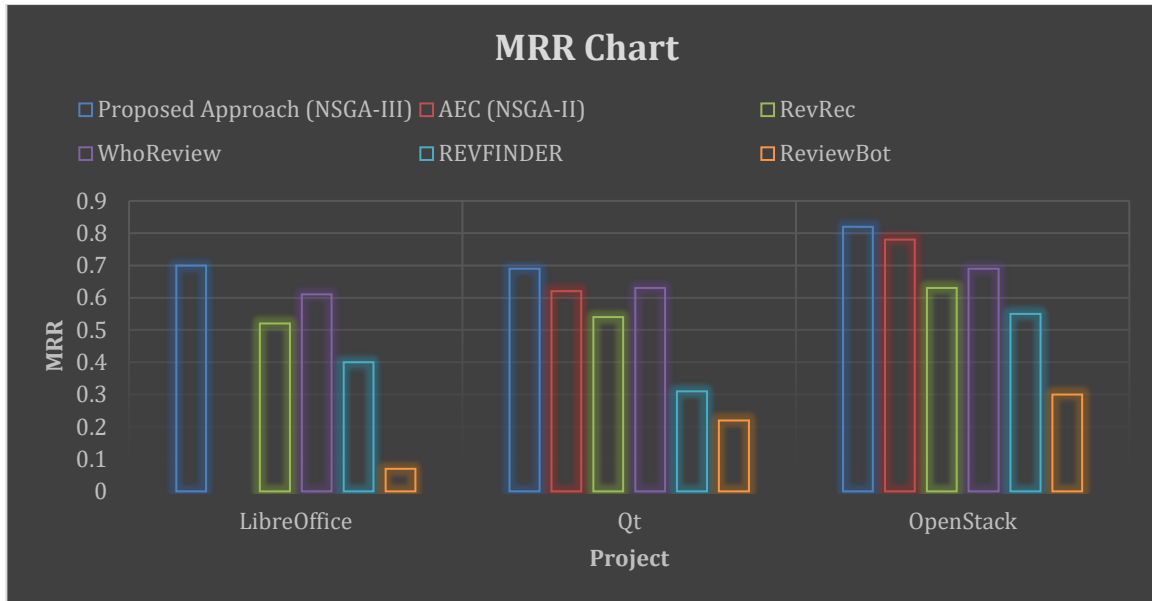


Figure 16. Comparison of MRR for Projects

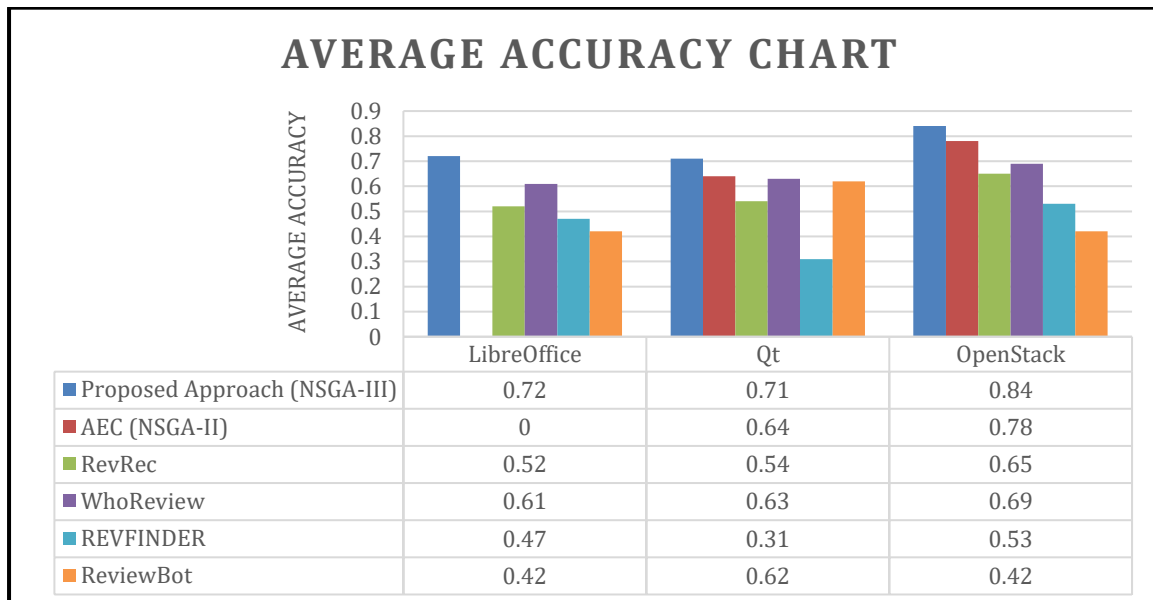


Figure 17. Average Accuracy of Projects

5.2.2 Answers to Research Questions

Answer for Question 1: The efficiency and precision to identify relevant code reviewers by using of our proposed approach i.e., NSGA-III is confirmed on pull requests from 3 different projects are resulted in table 15 & 16 and figure no 16 & 17. Table 15 and 16 shows the precision and recall in context to every experiment result separately. For example, LibreOffice has a precision ranging from 51% to 67% for all experiments. It has a recall range from 52% to 75%. Qt project has precision range of 50% to 69% and recall rate ranging from 58% to 73%. Whereas OpenStack project has the highest precision value of 80% and recall ranging from 69-86%. Due to large number of reviewers in the projects (i.e.,

Qt, LibreOffice) the precision rate of 50% or 51% in projects could also be considered acceptable. Also, some of the highest recall scores are obtained in OpenStack as some pull request require more than one code reviewer.

Figure 16 shows the MRR values that NSGA-III was able to efficiently rate the recommended code reviewers. The best resulted solution of the last population obtained in the last iteration of GA are copied in a single pool. Then, the rank of each reviewer corresponds to his frequency count in the pool. That is, reviewers that are recommended in many solutions are ranked first. The proposed approach shows the MRR values better than the all other presented in literature. LibreOffice came up with 70%, Qt with 69% and OpenStack with highest 82% MRR scores. The efficiency for ranking the reviewers is measured by this parameter so the outcomes of MRR are very important. Additionally, it is one of the main motivations of this proposed approach as ranking is directly related to availability and collaboration of code reviewers.

Figure 17 presents the average accuracy of our approach on experiments. The accuracy of each experiment is calculated separately and then average of accuracies is calculated as discussed in equations 9 and 10. LibreOffice came up with 72%, Qt with 71% and OpenStack with highest 84% Average Accuracy scores which are better than all others approach.

Most importantly our proposed technique doesn't have a bias towards the projects that are used for validation as we used average values of mean reciprocal rank and accuracy.

Out of 3 opensource projects, our proposed approach performed well on OpenStack pull requests. The accuracy, MRR, precision and recall of the project was good in comparison to other projects.

Answer for Question 2: While comparing the proposed approach performance, it is observed that the results, on average, are significantly better than the existing tools or frameworks (mono-objective approach of all objectives, search algorithms and other variants of GA) in terms Accuracy, Mean fitness and Execution time of algorithms. Figure 17 shows the comparison related to accuracy of our proposed approach with other approaches. The proposed approach clearly shows the highest accuracy in all 3 projects.

In context to comparison of the mean fitness and execution time with our proposed approach, we have selected NSGA-II [43] since this research was foremost and primarily relevant to our approach in terms of experimental sets and number of objectives. The authors [9] didn't provided any graphical representation of their approach, neither they provided any

comparison of execution time, thus we had to implement both approaches i.e., NSGA-II and NSGA-III to gather results.

Section 5.3 Provides graphical representation of framework using LibreOffice project from figure 18-35. Table 17, 18 and 19 gives mean fitness value comparison on the basis of each objective separately. Table 20 gives average execution time taken by both algorithms, and it can be clearly seen that NSGA-III takes less time as it does not require any additional adjustable parameters compare to NSGA-II [9].

Section 5.4 Provides graphical representation of framework using Qt project from figure 36-53. Table 21, 22 and 23 gives mean fitness value comparison on the basis of each objective separately. Table 24 gives average execution time taken by both algorithms, and it can be clearly seen that NSGA-III takes less time.

Section 5.5 Provides graphical representation of framework using OpenStack project from figure 54-71. Table 25, 26 and 27 gives mean fitness value comparison on the basis of each objective separately. Table 28 gives average execution time taken by both algorithms.

Figure 72 and 73 are a result to an additional check we did on this project by adding a new experimental set as Experiment no 7 in Table 14. Both the algorithms showed an abnormal behavior in terms of maximizing and minimizing the objectives resulting in premature convergence. The premature convergence of a genetic algorithm arises when the genes of some high rated individuals quickly attain to dominate the population, constraining it to converge to a local optimum. Also, when the generation size / termination criteria are very less than the pop size. The premature convergence is generally due to the loss of diversity within the population. Rest details of each graph and table is presented with them in description.

5.3 Comparison of Framework using LibreOffice Dataset

5.3.1 Comparison of visualization of ‘Availability’ (NSGA-II vs NSGA-III)

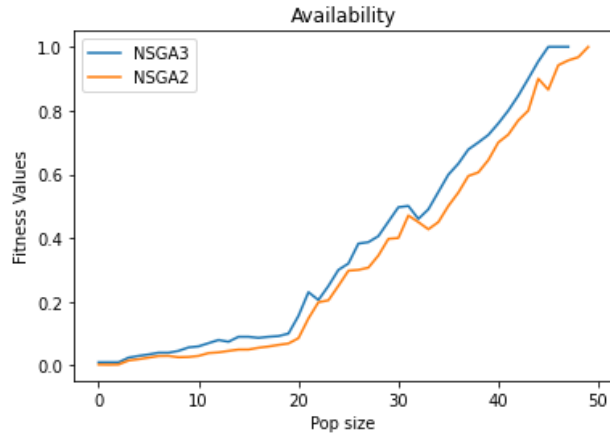


Figure 18. LibreOffice Availability Best Solution Experiment-1

The figure shows the fitness value against each chromosome in the population and the mean of all the fitness values. The mean fitness value for ‘Availability’ objective for experiment 1 in NSGA-II is 0.338605 and NSGA-III is 0.386167 as shown in Figure 18.

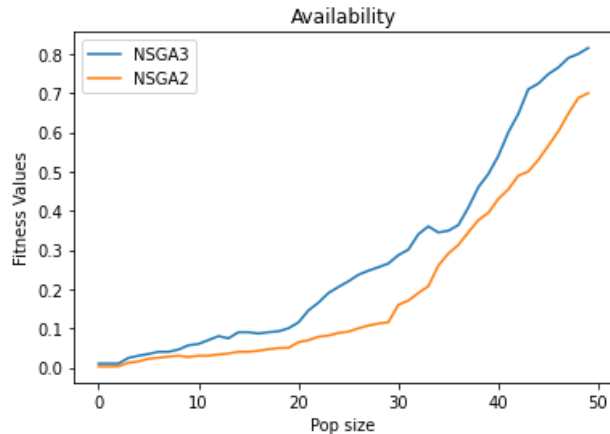


Figure 19. LibreOffice Availability Best Solution Experiment-2

The figure 19 shows the fitness value against each chromosome in the population and the mean of all the fitness values. The mean fitness value for ‘Availability’ objective for experiment 2 in NSGA-II is 0.195698 and NSGA-III is 0.280518.

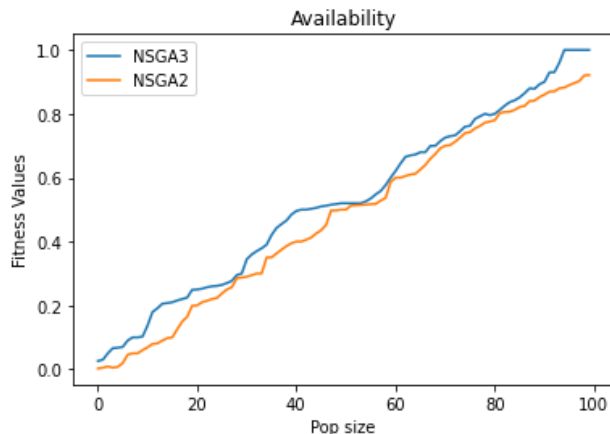


Figure 20. LibreOffice Availability Best Solution Experiment-3

The figure 20 shows the fitness value against each chromosome in the population and the mean of all the fitness values. The mean fitness value for ‘Availability’ objective for experiment 3 in NSGA-II is 0.476529 and NSGA-III is 0.526234.

Similarly, 21, 22 and 23 shows the fitness values against each chromosome in the population and the mean of all the fitness values. Their mean fitness values for ‘Availability’ objective are shared in table 17.

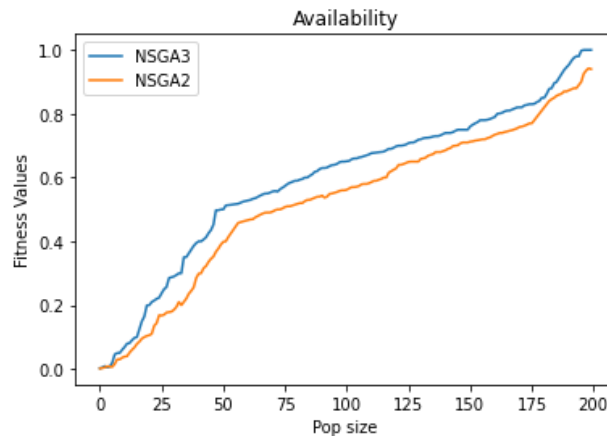


Figure 21. LibreOffice Availability Best Solution Experiment-4

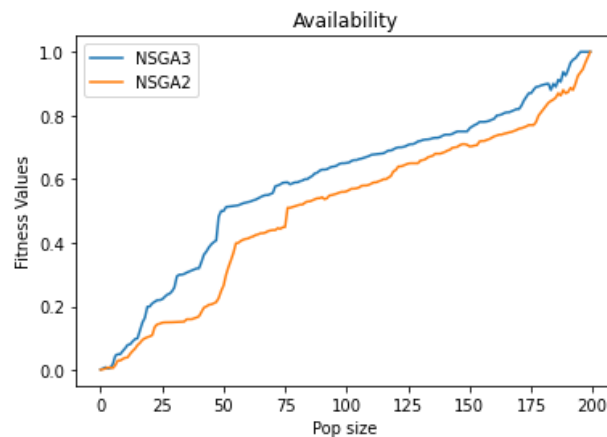


Figure 22. LibreOffice Availability Best Solution Experiment-5

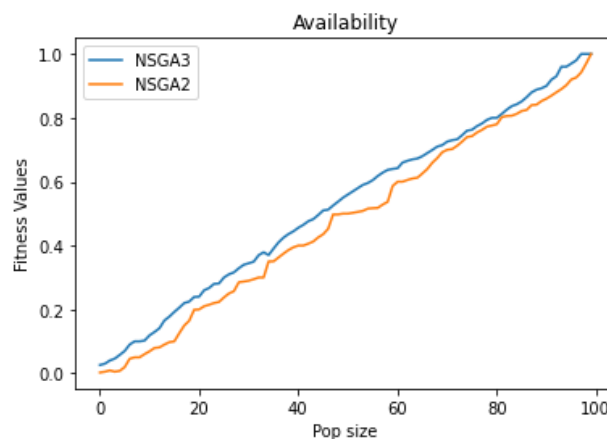


Figure 23. LibreOffice Availability Best Solution Experiment-6

Table 17. Mean Fitness of Availability (LibreOffice)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.338605	0.386167
Exp. 2	0.195698	0.280518
Exp. 3	0.476529	0.526234
Exp. 4	0.524646	0.593025
Exp. 5	0.506748	0.580662
Exp. 6	0.488974	0.548757

5.3.2 Comparison of visualization of ‘Collaboration’ (NSGA-II vs NSGA-III)

Figures 24-29 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Collaboration’ objective in LibreOffice project. Their mean fitness values are shared in table 18.

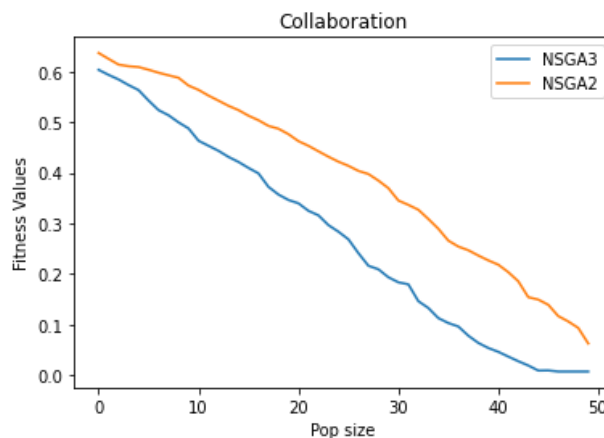


Figure 24. LibreOffice Collaboration Best Solution Experiment-1

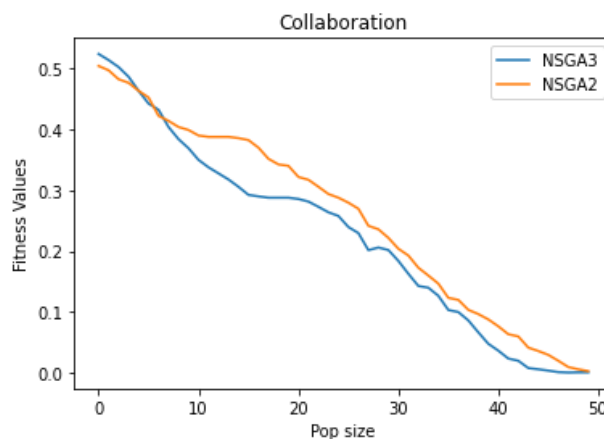


Figure 25. LibreOffice Collaboration Best Solution Experiment-2

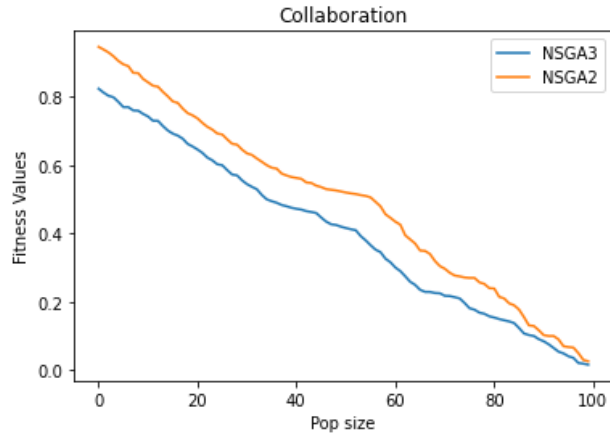


Figure 26. LibreOffice Collaboration Best Solution Experiment-3

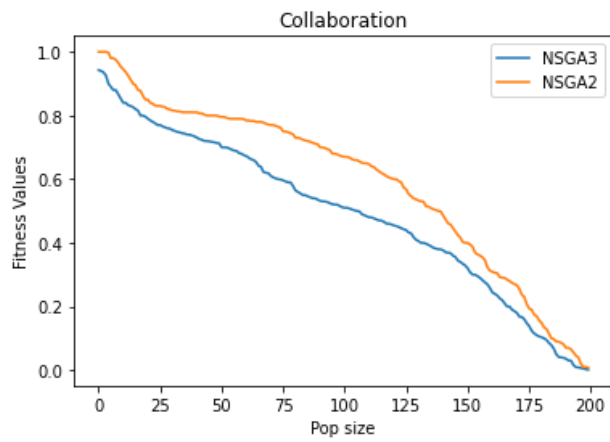


Figure 27. LibreOffice Collaboration Best Solution Experiment-4

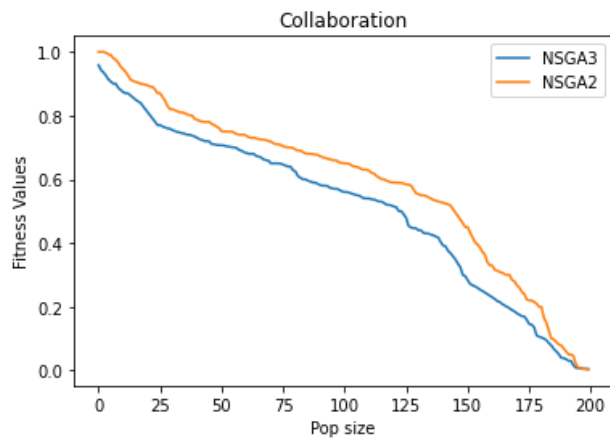


Figure 28. LibreOffice Collaboration Best Solution Experiment-5

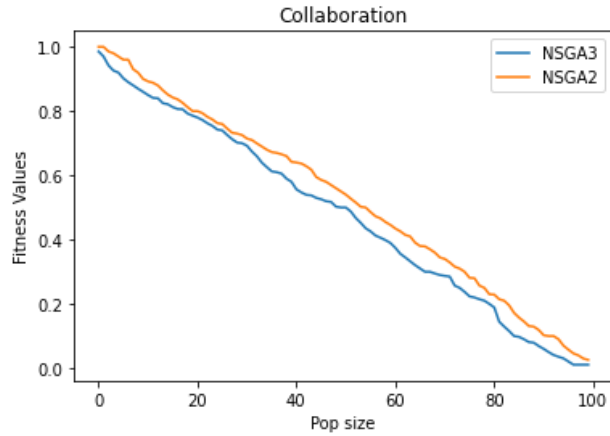


Figure 29. LibreOffice Collaboration Best Solution Experiment-6

Table 18. Mean Fitness of Collaboration (LibreOffice)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.393635	0.272019
Exp. 2	0.255101	0.215886
Exp. 3	0.490449	0.402135
Exp. 4	0.593836	0.492255
Exp. 5	0.590047	0.511345
Exp. 6	0.525513	0.476423

5.3.3 Comparison of visualization of ‘Expertise’ (NSGA-II vs NSGA-III)

Figures 30-36 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Expertise’ objective in LibreOffice project. Their mean fitness values are shared in table 19.

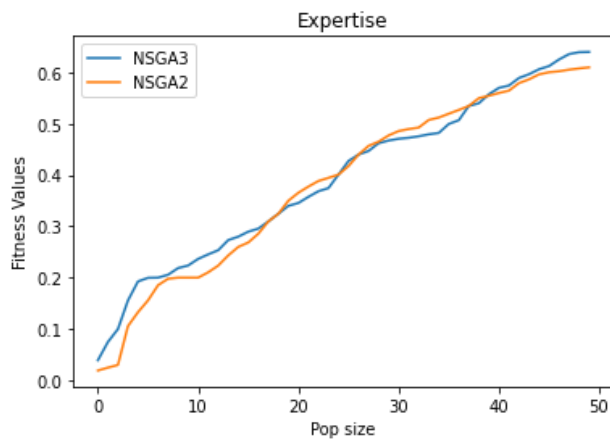


Figure 30. LibreOffice Expertise Best Solution Experiment-1

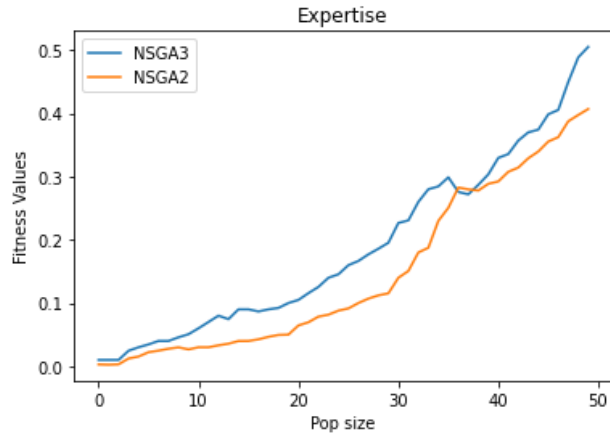


Figure 31. LibreOffice Expertise Best Solution Experiment-2

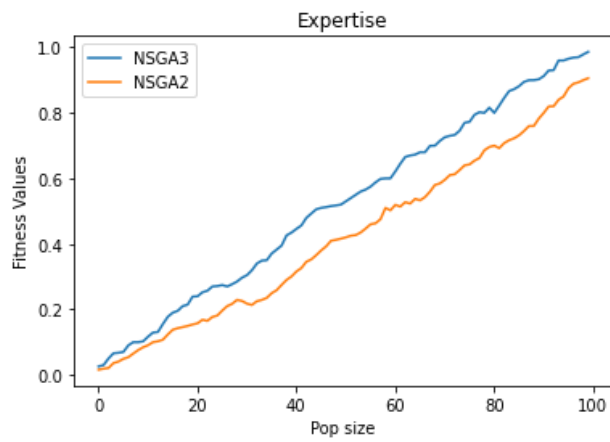


Figure 32. LibreOffice Expertise Best Solution Experiment-3

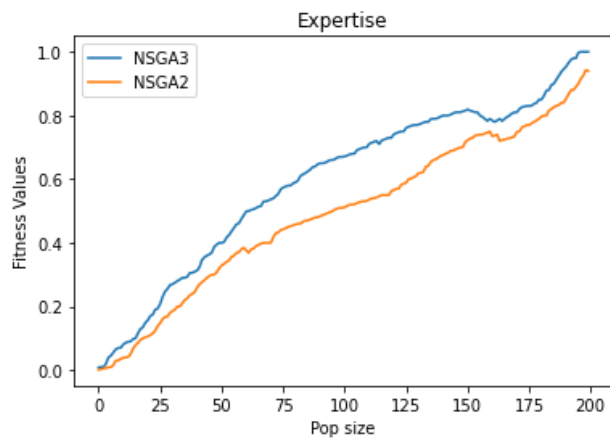


Figure 33. LibreOffice Expertise Best Solution Experiment-4

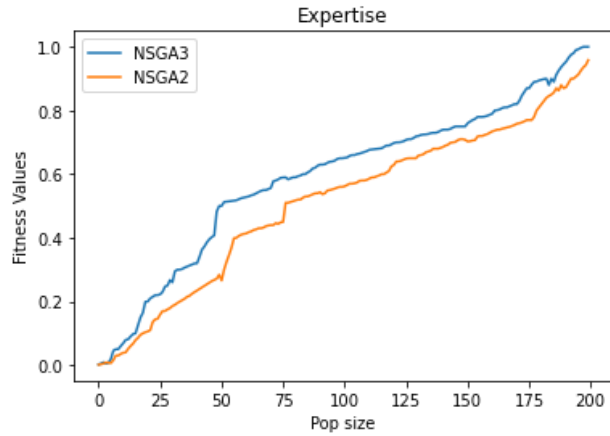


Figure 34. LibreOffice Expertise Best Solution Experiment-5

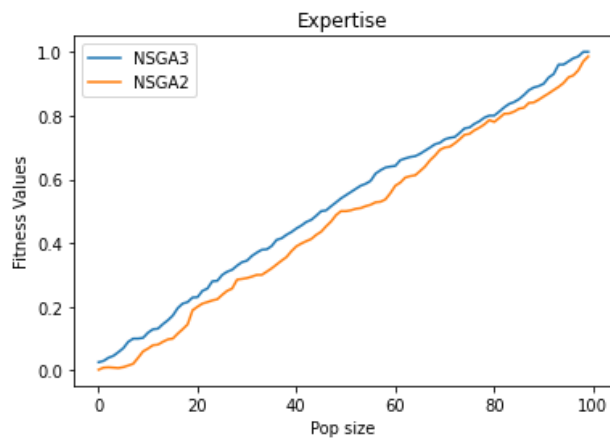


Figure 35. LibreOffice Expertise Best Solution Experiment-6

Table 19. Mean Fitness of Expertise (LibreOffice)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.383908	0.393141
Exp. 2	0.144642	0.187551
Exp. 3	0.421726	0.521721
Exp. 4	0.491682	0.593504
Exp. 5	0.511586	0.591058
Exp. 6	0.474786	0.525513

5.3.4 Comparison of execution time (NSGA-II vs NSGA-III)

Table 20. Comparison of Execution time for LibreOffice Project

Project Name	Experiment number	Execution Time	
		Proposed Approach (NSGA-III)	AEC (NSGA-II)
LibreOffice	Exp. 1	47 seconds	60 seconds
	Exp. 2	57 seconds	74 seconds
	Exp. 3	120 seconds	153 seconds

	Exp. 4	327 seconds	400 seconds
	Exp. 5	234 seconds	278 seconds
	Exp. 6	127 seconds	195 seconds

Table 20 shows the execution time comparison for all the objectives collectively for LibreOffice project. It can be clearly seen that NSGA-III takes less time to present the best solution in either objective.

5.4 Comparison of Framework using Qt Project

5.4.1 Comparison of visualization of ‘Availability’ (NSGA-II vs NSGA-III)

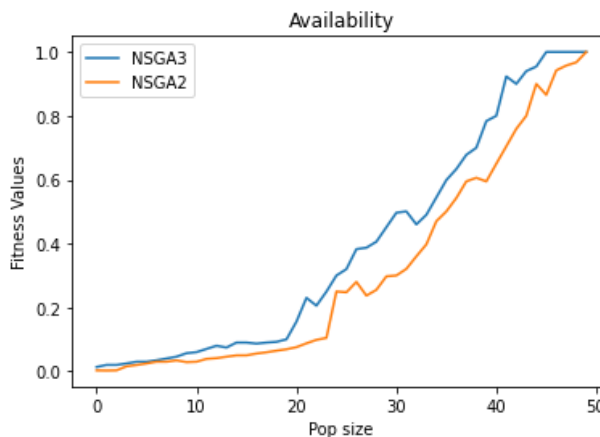


Figure 36. Qt Availability Best Solution Experiment-1

Figures 36-41 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Availability’ objective in Qt project. Their mean fitness values are shared in table 21. Figures 42-47 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Collaboration’ objective in Qt project. Their mean fitness values are shared in table 22. Figures 48-53 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Expertise’ objective in Qt project. Their mean fitness values are shared in table 23. Table 24 shows the execution time comparison for all the objectives collectively for Qt project. It can be clearly seen that NSGA-III takes less time to present the best solution in either objective.

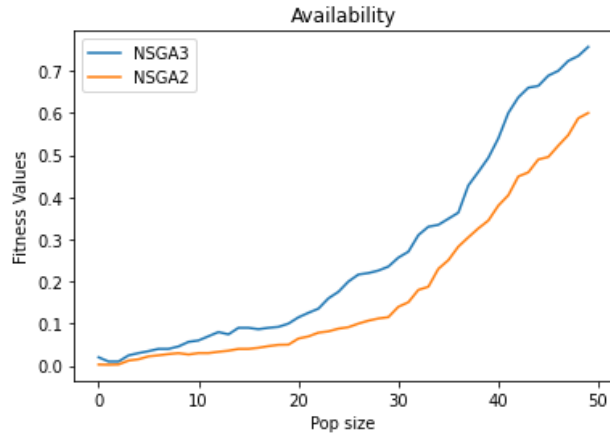


Figure 37. Qt Availability Best Solution Experiment-2

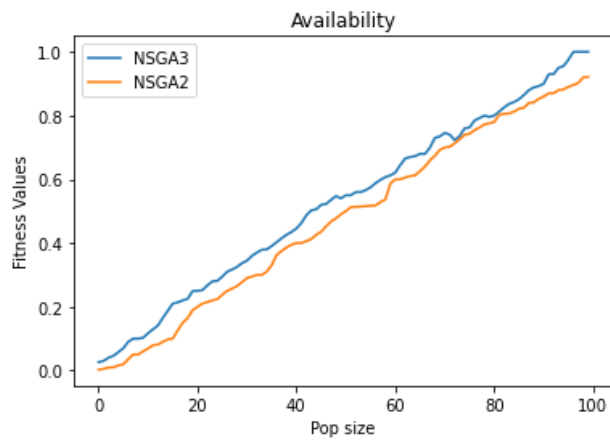


Figure 38. Qt Availability Best Solution Experiment-3

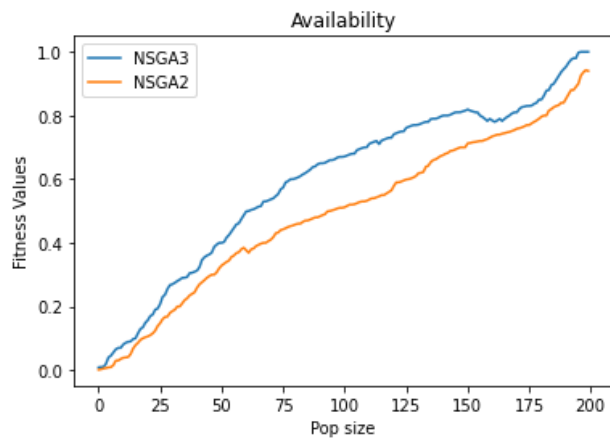


Figure 39. Qt Availability Best Solution Experiment-4

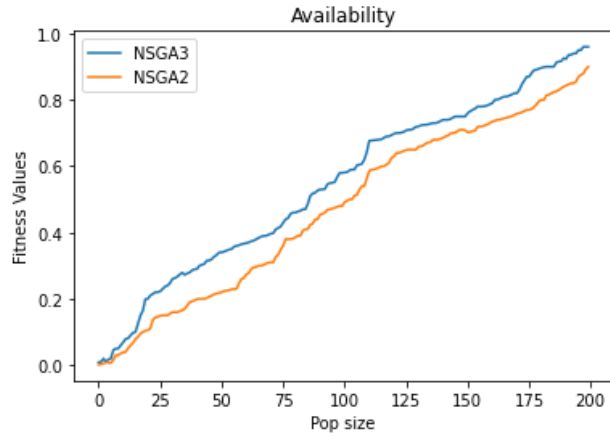


Figure 40. Qt Availability Best Solution Experiment-5

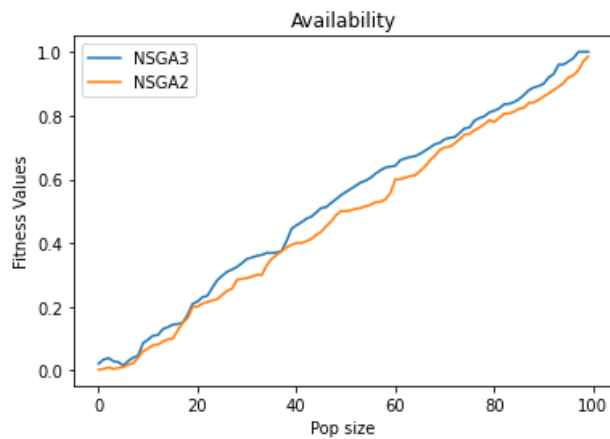


Figure 41. Qt Availability Best Solution Experiment-6

Table 21. Mean Fitness of Availability (Qt)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.317225	0.392775
Exp. 2	0.176078	0.265095
Exp. 3	0.474959	0.527416
Exp. 4	0.492255	0.593868
Exp. 5	0.471061	0.546245
Exp. 6	0.477343	0.519724

5.4.2 Comparison of visualization of ‘Collaboration’ (NSGA-II vs NSGA-III)

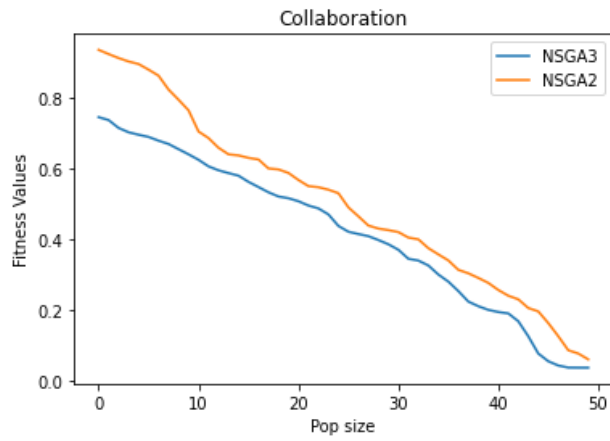


Figure 42. Qt Collaboration Best Solution Experiment-1

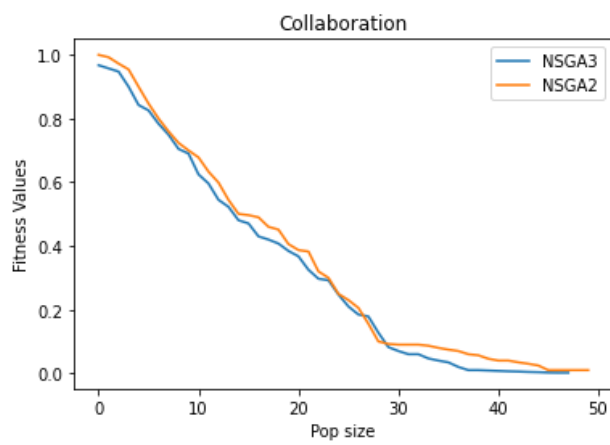


Figure 43. Qt Collaboration Best Solution Experiment-2

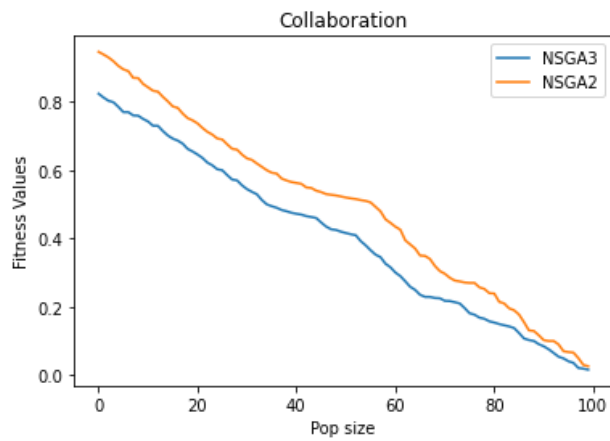


Figure 44. Qt Collaboration Best Solution Experiment-3

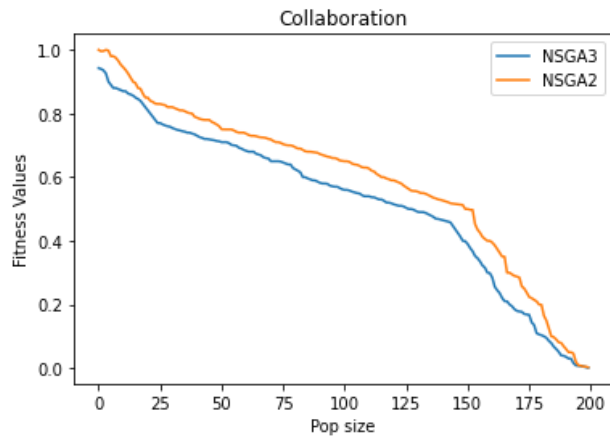


Figure 45. Qt Collaboration Best Solution Experiment-4

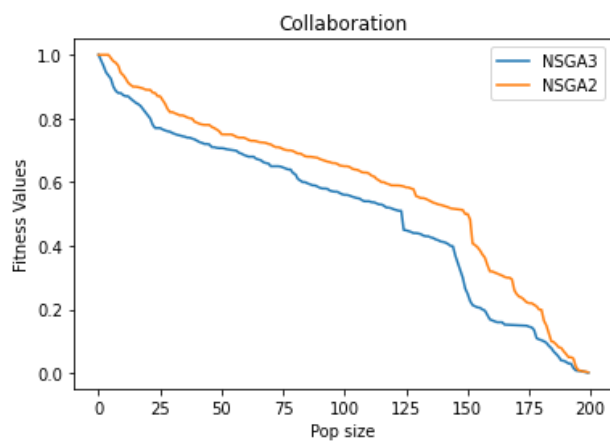


Figure 46. Qt Collaboration Best Solution Experiment-5

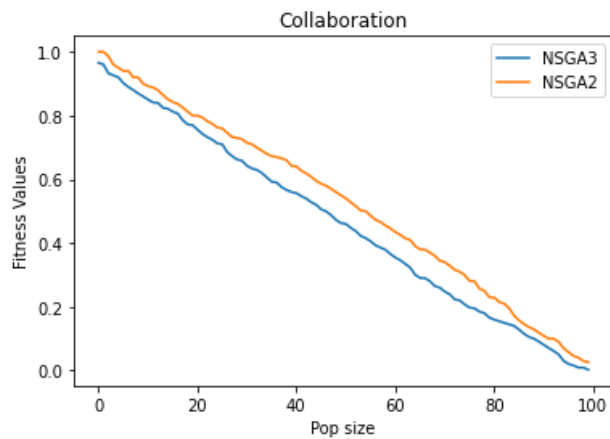


Figure 47. Qt Collaboration Best Solution Experiment-6

Table 22. Mean Fitness of Collaboration (Qt)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.503183	0.416552
Exp. 2	0.345868	0.332376
Exp. 3	0.490449	0.402135
Exp. 4	0.592985	0.524646

Exp. 5	0.590662	0.506748
Exp. 6	0.524423	0.462726

5.4.3 Comparison of visualization of ‘Expertise’ (NSGA-II vs NSGA-III)

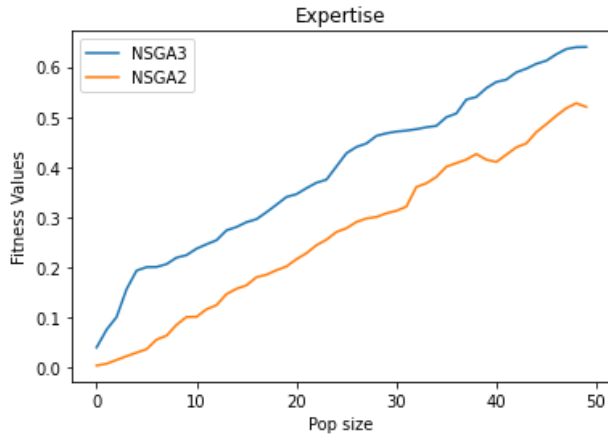


Figure 48. Qt Expertise Best Solution Experiment-1

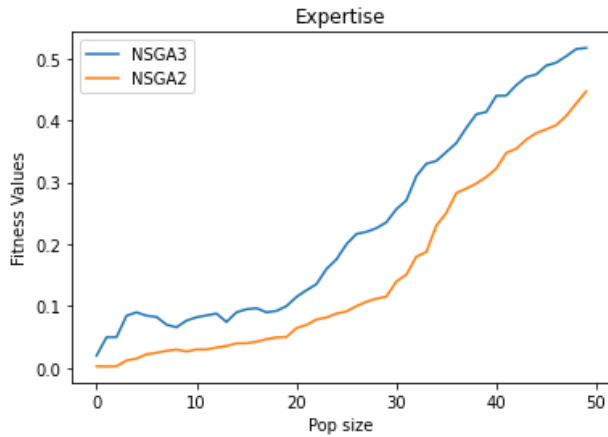


Figure 49. Qt Expertise Best Solution Experiment-2

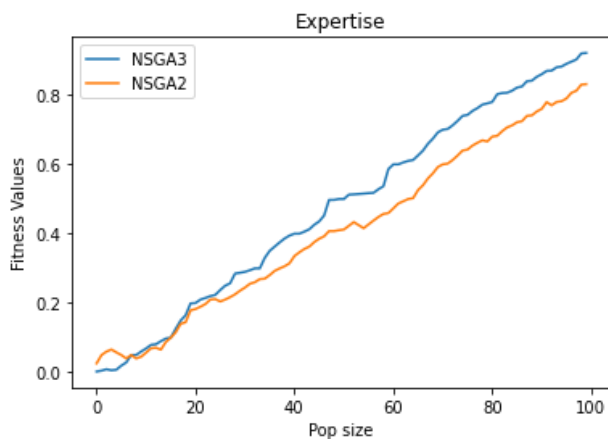


Figure 50. Qt Expertise Best Solution Experiment-3

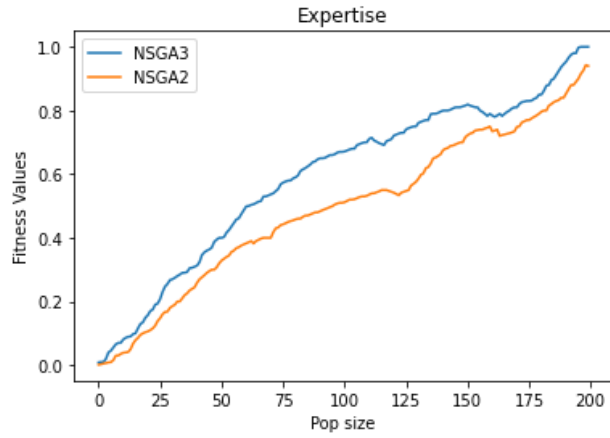


Figure 51. Qt Expertise Best Solution Experiment-4

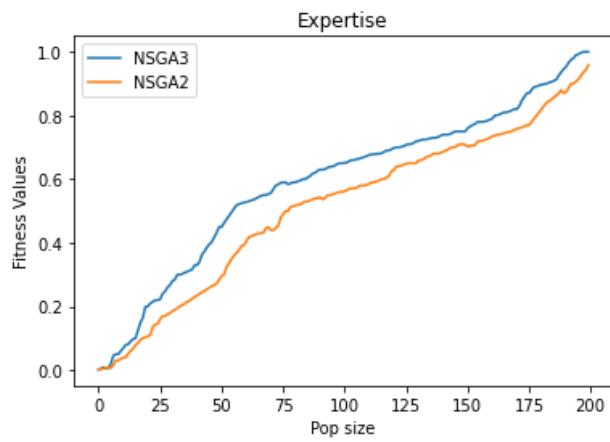


Figure 52. Qt Expertise Best Solution Experiment-5

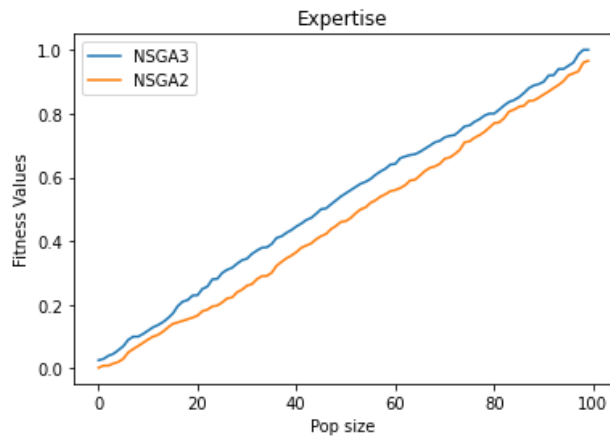


Figure 53. Qt Expertise Best Solution Experiment-6

Table 23. Mean Fitness of Expertise (Qt)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.264076	0.393145
Exp. 2	0.152442	0.232004
Exp. 3	0.412261	0.476169
Exp. 4	0.488958	0.591406

Exp. 5	0.511318	0.599904
Exp. 6	0.462726	0.524423

5.4.4 Comparison of execution time (NSGA-II vs NSGA-III)

Table 24. Comparison of Execution time for Qt Project

Project Name	Experiment number	Execution Time	
		Proposed Approach (NSGA-III)	AEC (NSGA-II)
Qt	Exp. 1	21 seconds	34 seconds
	Exp. 2	16 seconds	22 seconds
	Exp. 3	104 seconds	112 seconds
	Exp. 4	235 seconds	249 seconds
	Exp. 5	304 seconds	320 seconds
	Exp. 6	120 seconds	148 seconds

5.5 Comparison of Framework using OpenStack Project

5.5.1 Comparison of visualization of ‘Availability’ (NSGA-II vs NSGA-III)

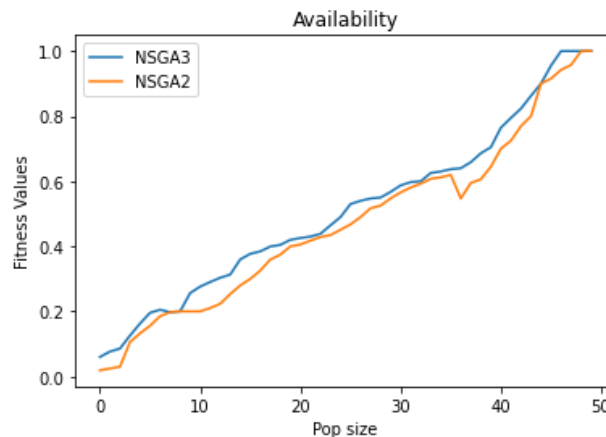


Figure 54. OpenStack Availability Best Solution Experiment-1

Figures 54-59 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Availability’ objective in OpenStack project. Their mean fitness values are shared in table 25. Figures 60-65 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Collaboration’ objective. Their mean fitness values are shared in table 26. Figures 66-71 shows the fitness values against each chromosome in the population and the mean of all the fitness values for ‘Expertise’ objective. Their mean fitness values are shared in table 27. Table 28 shows the execution time comparison for all the objectives collectively for Qt project. It can be clearly seen that NSGA-III takes less time to present the best solution in either objective.

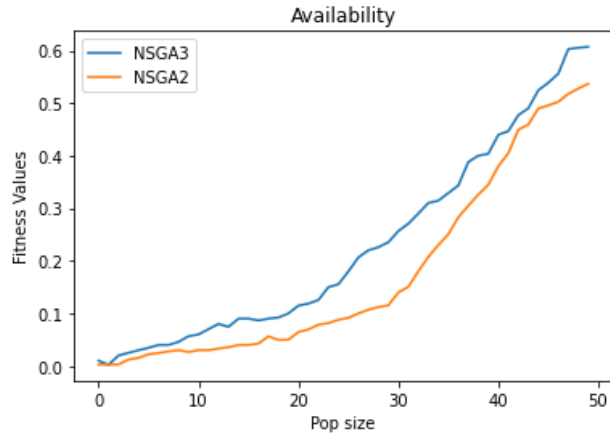


Figure 55. OpenStack Availability Best Solution Experiment-2

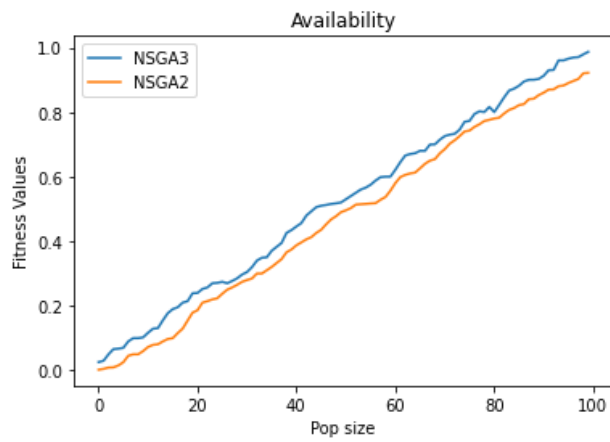


Figure 56. OpenStack Availability Best Solution Experiment-3

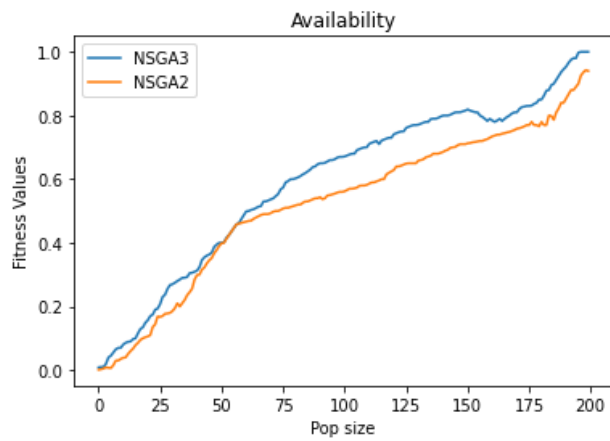


Figure 57. OpenStack Availability Best Solution Experiment-4

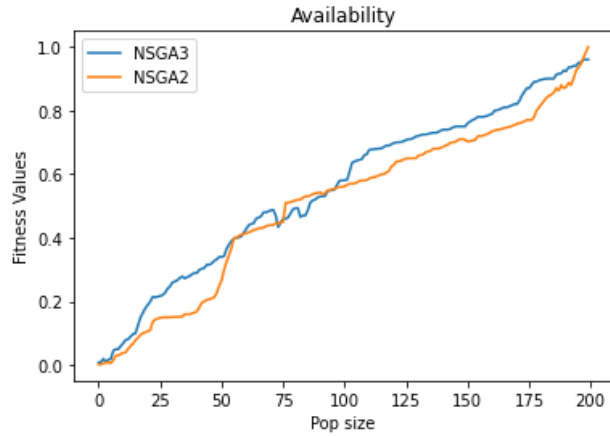


Figure 58. OpenStack Availability Best Solution Experiment-5

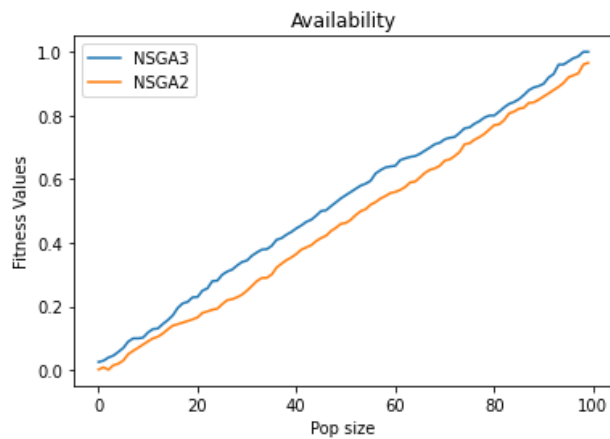


Figure 59. OpenStack Availability Best Solution Experiment-6

Table 25. Mean Fitness of Availability (OpenStack)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.470803	0.510869
Exp. 2	0.173214	0.229236
Exp. 3	0.471429	0.521721
Exp. 4	0.521591	0.593878
Exp. 5	0.506748	0.555093
Exp. 6	0.462262	0.525513

5.5.2 Comparison of visualization of ‘Collaboration’ (NSGA-II vs NSGA-III)

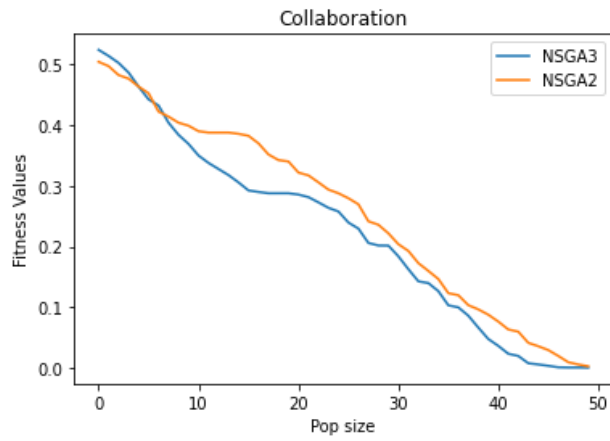


Figure 60. OpenStack Collaboration Best Solution Experiment-1

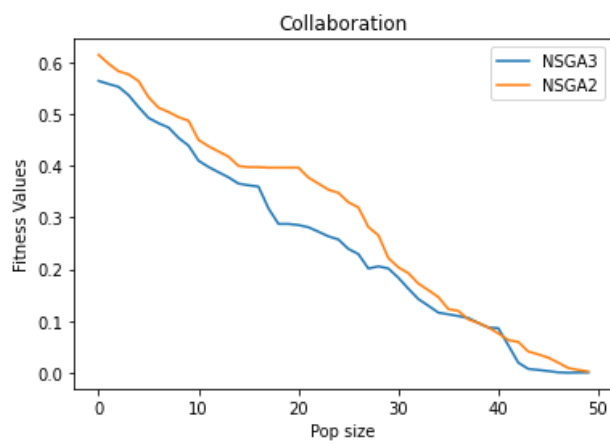


Figure 61. OpenStack Collaboration Best Solution Experiment-2

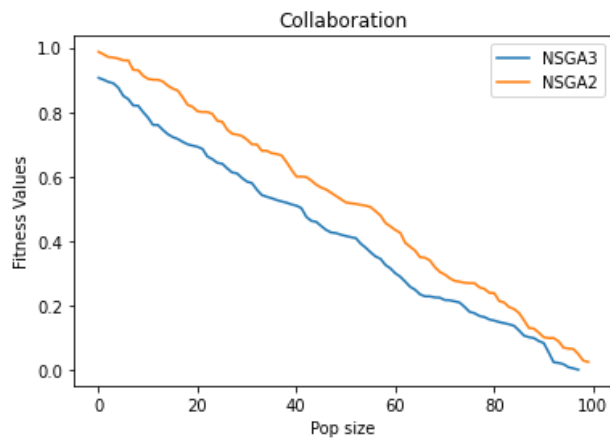


Figure 62. OpenStack Collaboration Best Solution Experiment-3

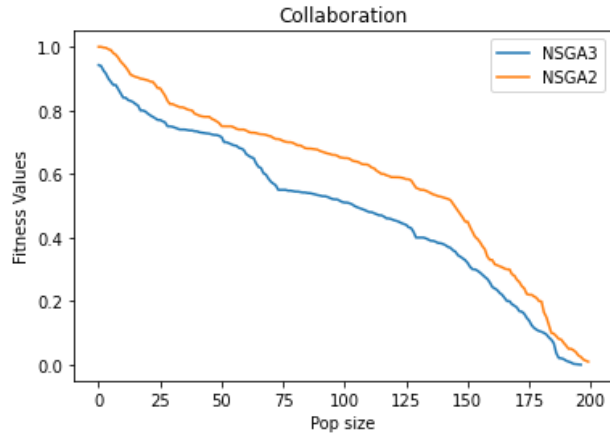


Figure 63. OpenStack Collaboration Best Solution Experiment-4

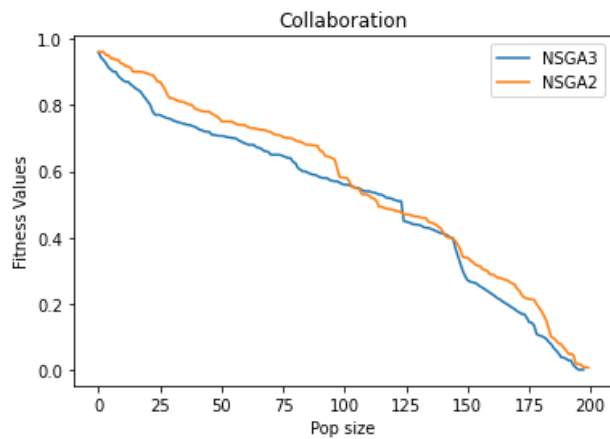


Figure 64. OpenStack Collaboration Best Solution Experiment-5

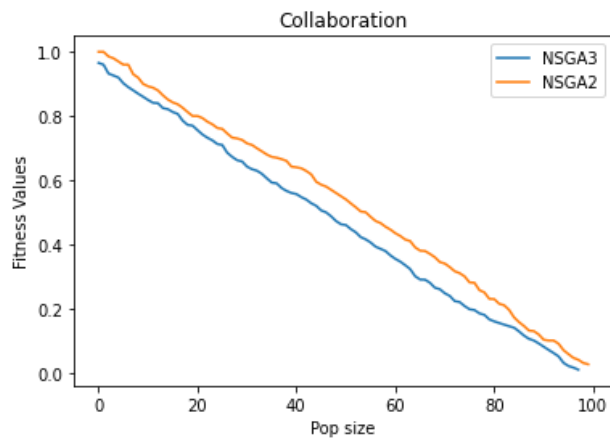


Figure 65. OpenStack Collaboration Best Solution Experiment-6

Table 26. Mean Fitness of Collaboration (OpenStack)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.255101	0.215886
Exp. 2	0.291583	0.249566
Exp. 3	0.521721	0.427924
Exp. 4	0.620451	0.495459

Exp. 5	0.555093	0.516667
Exp. 6	0.525513	0.452054

5.5.3 Comparison of visualization of ‘Expertise’ (NSGA-II vs NSGA-III)

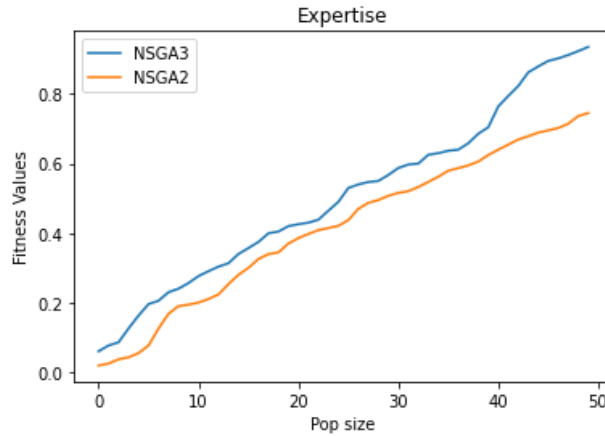


Figure 66. OpenStack Expertise Best Solution Experiment-1

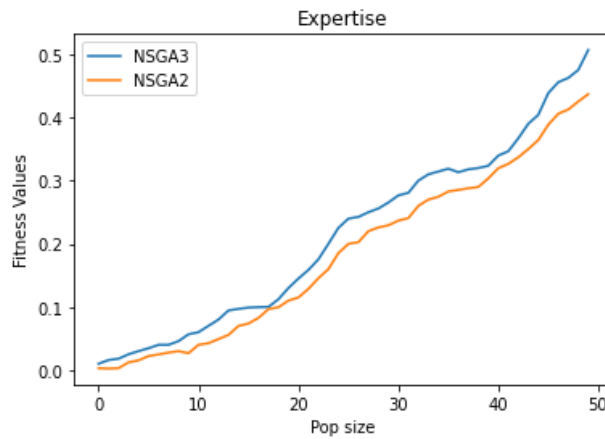


Figure 67. OpenStack Expertise Best Solution Experiment-2

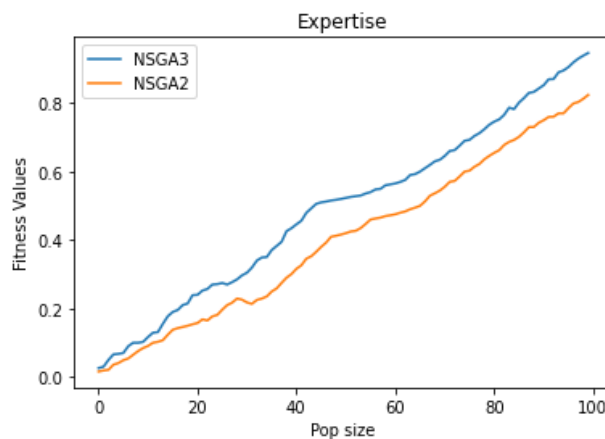


Figure 68. OpenStack Expertise Best Solution Experiment-3

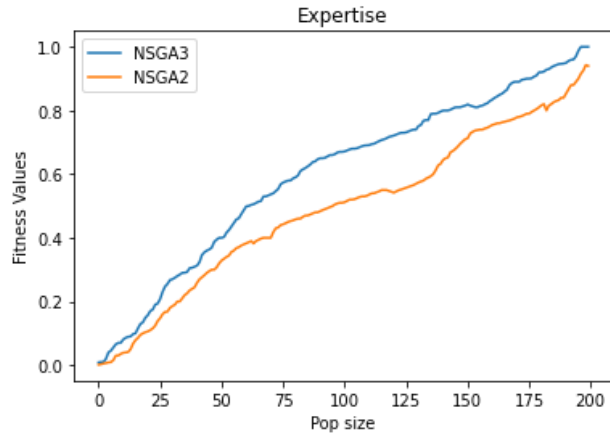


Figure 69. OpenStack Expertise Best Solution Experiment-4

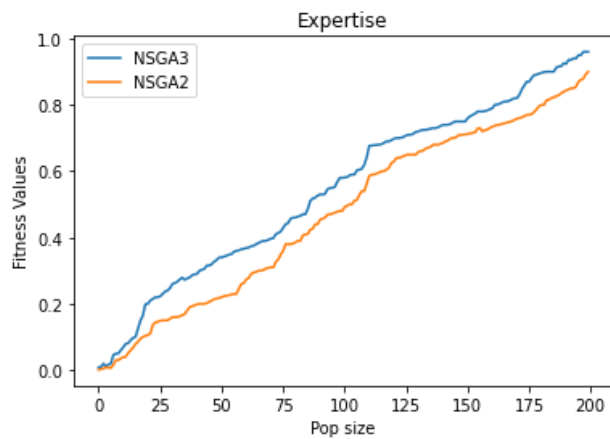


Figure 70. OpenStack Expertise Best Solution Experiment-5

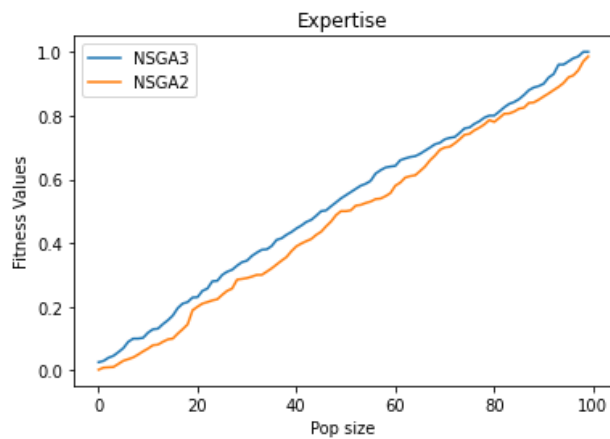


Figure 71. OpenStack Expertise Best Solution Experiment-6

Table 27. Mean Fitness of Expertise (OpenStack)

Mean Fitness:

Experiment Number	NSGA-II	NSGA-III
Exp. 1	0.415962	0.503183
Exp. 2	0.183926	0.213556
Exp. 3	0.402135	0.490449
Exp. 4	0.489821	0.600051

Exp. 5	0.471376	0.546245
Exp. 6	0.476325	0.525513

5.5.4 Comparison of execution time (NSGA-II vs NSGA-III)

Table 28. Comparison of Execution time for OpenStack Project

Project Name	Experiment number	Execution Time	
		Proposed Approach (NSGA-III)	AEC (NSGA-II)
OpenStack	Exp. 1	18 seconds	26 seconds
	Exp. 2	24 seconds	32 seconds
	Exp. 3	152 seconds	159 seconds
	Exp. 4	234 seconds	249 seconds
	Exp. 5	246 seconds	282 seconds
	Exp. 6	119 seconds	129 seconds

5.5.5 Abnormal / Premature Behavior of Algorithm (NSGA-II vs NSGA-III) for Experiment no 7 (OpenStack Project)

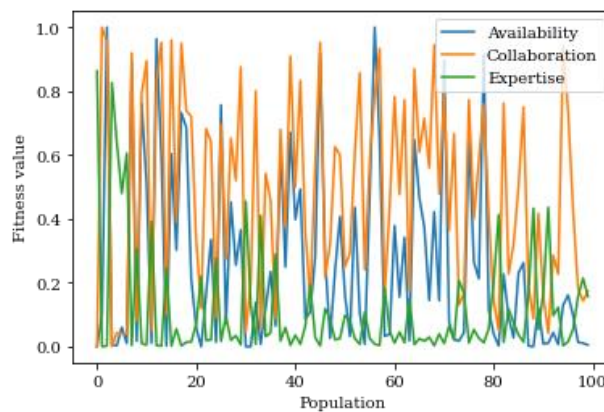


Figure 72. LibreOffice Abnormal / Premature Behavior of Algorithm NSGA-II

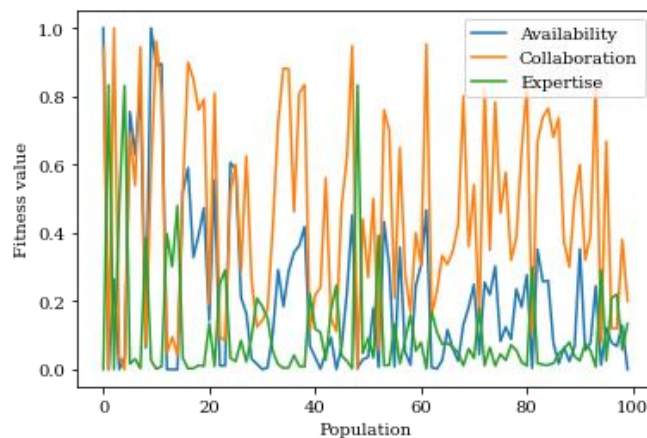


Figure 73. LibreOffice Abnormal / Premature Behavior of Algorithm NSGA-III

Figure 72 and 73 are a result to Experiment no 7 in Table 14. Both the algorithms NSGA-II and NSGA-III showed an abnormal behavior in terms of maximizing and minimizing the objectives resulting in premature convergence. The premature convergence of a genetic algorithm arises when the genes of some high rated individuals quickly attain to dominate the population, constraining it to converge to a local optimum [44]. Also, when the generation size / termination criteria are very less than the pop size. The premature convergence is generally due to the loss of diversity within the population [45].

5.6 Limitations

The results i.e., graphical representations, precision, recall, accuracy and MRR all are calculated on the basis of number of generations i.e., 500, as we have to compare our results with the literature [9], [11], [51], [69]. Moreover, we only added one new experiment set and remaining were used from the same literature experimental set-in change operators' parameters for comparison purposes. All the experimental set values used were selected by trial-and-error method.

The definition of expertise and collaborations can be subjective and hard to formalize thus further empirical studies are required to validate the different metrics used in our work. We are planning to consider other possible formations as part of our future work and compare between them. For example: our current definition of the availability needs further improvement. Like, reviewers can be assigned other types of development activities than coding (e.g., testing, design/architecture, requirements analysis, etc.). The data about these activities are not always available. However, the formulation of our fitness function is easy to modify in a way that enables managers to enter the number of tasks per reviewer, especially the ones that they are beyond code reviews.

CONCLUSION & FUTURE WORK

Chapter 6 : CONCLUSION & FUTURE WORK

6.1 Conclusion

In this research, we have proposed a multi objectives problem to manage and recommend code reviewers by adopting an optimization algorithm that is NSGA-III. The purpose is to recommend the best trade-off reviewers between three conflicting objectives i.e., maximizing the availability and expertise of reviewers and minimizing the collaboration between developers and reviewers to lessen the human biasness factor. We implemented and evaluated our approach on three (medium to large size) open-source projects named as LibreOffice, Qt and OpenStack. We calculated efficiency on our approach by finding precision, recall, MRR, accuracy for all 3 projects on average. The results from our proposed approach accurately recommended the code reviewers with the precision up to 80%, 86% of recall, 82% mean reciprocal rank and 84% average accuracy by improving state-of-the-art. Then we implemented NSGA-III and NSGA-II in terms of finding and comparing mean fitness and execution time of both algorithms while keeping the objective and experimental sets same. As a result, NSGA-III recommended the reviewers in less execution time and better mean fitness values in comparison to NSGA-II in all experimental sets. NSGA-III algorithm was able to find a well-converged and well-distributed set of solutions. The proposed approach could be practical to MCR in order to help developers while recommending suitable code-reviewers in less time and resources to speed up the review process. This research highlighted the importance of managing code reviews to reduce delays in review process in less time and resources while confirming high expertise and availability as much as possible

6.2 Future Work

As future work, the proposed approach can be extended to implement by increasing the number of objectives i.e., the size of code change, reviews recency and quality in the past etc. Also, the use of more projects can be added to test the approach.

REFERENCES

- [1] H. A. Çetin, E. Doğan, and E. J. S. o. C. P. Tüzün, "A review of code reviewer recommendation studies: Challenges and future directions," vol. 208, p. 102652, 2021.
- [2] M. B. Zanjani, H. Kagdi, and C. J. I. T. o. S. E. Bird, "Automatically recommending peer reviewers in modern code review," vol. 42, no. 6, pp. 530-543, 2015.
- [3] A. Wahid, X. Gao, and P. Andreae, "Multi-view clustering of web documents using multi-objective genetic algorithm," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 2625-2632: IEEE.
- [4] C. Thompson and D. Wagner, "A large-scale study of modern code review and security in open source projects," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 83-92.
- [5] A. Ouni *et al.*, "Search-based software library recommendation using multi-objective optimization," vol. 83, pp. 55-75, 2017.
- [6] K. Hamasaki, R. G. Kula, N. Yoshida, A. C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 49-52: IEEE.
- [7] Y. Yu, H. Wang, G. Yin, T. J. I. Wang, and S. Technology, "Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?," vol. 74, pp. 204-218, 2016.
- [8] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 192-201.
- [9] S. Rebai, A. Amich, S. Molaei, M. Kessentini, and R. J. A. S. E. Kazman, "Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations," vol. 27, no. 3, pp. 301-328, 2020.
- [10] R. H. Bhesdadiya, I. N. Trivedi, P. Jangir, N. Jangir, and A. J. C. E. Kumar, "An NSGA-III algorithm for solving multi-objective economic/environmental dispatch problem," vol. 3, no. 1, p. 1269383, 2016.
- [11] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "Who should review my code? a file location-based code-reviewer

- recommendation approach for modern code review," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 141-150: IEEE.
- [12] R. Almhana, W. Mkaouer, M. Kessentini, and A. Ouni, "Recommending relevant classes for bug reports using multi-objective search," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 286-295: IEEE.
- [13] X. Zhang *et al.*, "How do multiple pull requests change the same code: A study of competing pull requests in github," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 228-239: IEEE.
- [14] J. Jiang, A. Mohamed, and L. J. I. A. Zhang, "What are the characteristics of reopened pull requests? a case study on open source projects in github," vol. 7, pp. 102751-102761, 2019.
- [15] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu, "A hybrid approach to code reviewer recommendation with collaborative filtering," in *2017 6th International Workshop on Software Mining (SoftwareMining)*, 2017, pp. 24-31: IEEE.
- [16] Y. Hu, J. Wang, J. Hou, S. Li, and Q. Wang, "Is There A "Golden" Rule for Code Reviewer Recommendation?:—An Experimental Evaluation," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 2020, pp. 497-508: IEEE.
- [17] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2014, pp. 119-122.
- [18] J. K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, "Core: Automating review recommendation for code changes," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 284-295: IEEE.
- [19] K. Deb, "Multi-objective optimization," in *Search methodologies*: Springer, 2014, pp. 403-449.
- [20] S. Mirjalili, "Genetic algorithm," in *Evolutionary algorithms and neural networks*: Springer, 2019, pp. 43-55.
- [21] S. Katoch, S. S. Chauhan, V. J. M. T. Kumar, and Applications, "A review on genetic algorithm: past, present, and future," vol. 80, no. 5, pp. 8091-8126, 2021.

- [22] B. Kitchenham *et al.*, "Systematic literature reviews in software engineering—a systematic literature review," vol. 51, no. 1, pp. 7-15, 2009.
- [23] E. Sülün, E. Tüzün, and U. Doğrusöz, "Reviewer recommendation using software artifact traceability graphs," in *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, 2019, pp. 66-75.
- [24] A. Ouni, R. G. Kula, and K. Inoue, "Search-based peer reviewers recommendation in modern code review," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 367-377: IEEE.
- [25] M. M. Rahman, C. K. Roy, and J. A. Collins, "Correct: code reviewer recommendation in github based on cross-project and technology experience," in *Proceedings of the 38th international conference on software engineering companion*, 2016, pp. 222-231.
- [26] N. Sadman, M. M. Ahsan, and M. P. Mahmud, "ADCR: An Adaptive Tool to select” Appropriate Developer for Code Review” based on Code Context," in *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2020, pp. 0583-0591: IEEE.
- [27] S. Rostami and A. Shenfield, "Cma-paes: Pareto archived evolution strategy using covariance matrix adaptation for multi-objective optimisation," in *2012 12th UK Workshop on Computational Intelligence (UKCI)*, 2012, pp. 1-8: IEEE.
- [28] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, "Performance comparison of NSGA-II and NSGA-III on various many-objective test problems," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 3045-3052: IEEE.
- [29] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 931-940: IEEE.
- [30] X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change?: Putting text and file location analyses together for more accurate recommendations," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 261-270: IEEE.
- [31] M. M. Rahman, C. K. Roy, and R. G. Kula, "Predicting usefulness of code review comments using textual features and developer experience," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 215-226: IEEE.

- [32] E. Doğan, E. Tüzün, K. A. Tecimer, and H. A. Güvenir, "Investigating the validity of ground truth in code reviewer recommendation studies," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1-6: IEEE.
- [33] J. Lipcak and B. Rossi, "A large-scale study on source code reviewer recommendation," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018, pp. 378-387: IEEE.
- [34] M. Mukadam, C. Bird, and P. C. Rigby, "Gerrit software code review data from android," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 45-48: IEEE.
- [35] R. Geng, J. Zhou, and S. Yang, "MSOPS-IIA: An Accelerated Version of MSOPS-II," in *2015 8th International Symposium on Computational Intelligence and Design (ISCID)*, 2015, vol. 2, pp. 327-331: IEEE.
- [36] Y. Tian, X. Zhang, R. Cheng, and Y. Jin, "A multi-objective evolutionary algorithm based on an enhanced inverted generational distance metric," in *2016 IEEE congress on evolutionary computation (CEC)*, 2016, pp. 5222-5229: IEEE.
- [37] A. S. Akopov and M. A. Hevencev, "A multi-agent genetic algorithm for multi-objective optimization," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 1391-1395: IEEE.
- [38] M. Z. Rafique, K. A. Alam, and U. Iqbal, "Multi-Objective Optimization Techniques for Software Refactoring: A Systematic Literature Review," in *2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*, 2019, pp. 1-7: IEEE.
- [39] S. Zhang, H. Wang, D. Yang, and M. Huang, "Hybrid multi-objective genetic algorithm for multi-objective optimization problems," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, 2015, pp. 1970-1974: IEEE.
- [40] S. Fuquan, W. Hongfeng, and L. Fuqiang, "A species-based multi-objective genetic algorithm for multi-objective optimization problems," in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, 2014, pp. 5063-5066: IEEE.
- [41] D. H. Phan and J. Suzuki, "R2-IBEA: R2 indicator based evolutionary algorithm for multiobjective optimization," in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 1836-1845: IEEE.
- [42] Z. He and G. G. Yen, "Diversity improvement in decomposition-based multi-objective evolutionary algorithm for many-objective optimization problems," in *2014*

- IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 2409-2414: IEEE.
- [43] Y. Tian, R. Cheng, X. Zhang, and Y. J. I. C. I. M. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum]," vol. 12, no. 4, pp. 73-87, 2017.
- [44] K. Deb and H. J. I. t. o. e. c. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints," vol. 18, no. 4, pp. 577-601, 2013.
- [45] H. Jain and K. J. I. T. o. e. c. Deb, "An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach," vol. 18, no. 4, pp. 602-622, 2013.
- [46] R. Cheng, Y. Jin, M. Olhofer, and B. J. I. T. o. E. C. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," vol. 20, no. 5, pp. 773-791, 2016.
- [47] Y. Tian, R. Cheng, X. Zhang, F. Cheng, and Y. J. I. T. o. E. C. Jin, "An indicator-based multiobjective evolutionary algorithm with reference point adaptation for better versatility," vol. 22, no. 4, pp. 609-622, 2017.
- [48] K. Sindhya, K. Miettinen, and K. J. I. T. o. E. C. Deb, "A hybrid framework for evolutionary multi-objective optimization," vol. 17, no. 4, pp. 495-511, 2012.
- [49] S. Jiang and S. J. I. T. o. E. C. Yang, "A strength Pareto evolutionary algorithm based on reference direction for multiobjective and many-objective optimization," vol. 21, no. 3, pp. 329-346, 2017.
- [50] Y. Sun, G. G. Yen, and Z. J. I. T. o. E. C. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," vol. 23, no. 2, pp. 173-187, 2018.
- [51] M. Chouchen, A. Ouni, M. W. Mkaouer, R. G. Kula, and K. J. A. S. C. Inoue, "WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review," vol. 100, p. 106908, 2021.
- [52] J. Jiang, Y. Yang, J. He, X. Blanc, L. J. I. Zhang, and S. Technology, "Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development," vol. 84, pp. 48-62, 2017.
- [53] E. Sülün, E. Tüzün, U. J. I. Doğrusöz, and S. Technology, "RSTrace+: Reviewer suggestion using software artifact traceability graphs," vol. 130, p. 106455, 2021.

- [54] F. Li, R. Cheng, J. Liu, and Y. J. A. S. C. Jin, "A two-stage R2 indicator based evolutionary algorithm for many-objective optimization," vol. 67, pp. 245-260, 2018.
- [55] L. Cui, P. Ou, X. Fu, Z. Wen, N. J. J. o. P. Lu, and D. Computing, "A novel multi-objective evolutionary algorithm for recommendation systems," vol. 103, pp. 53-63, 2017.
- [56] H. Afshari, W. Hare, and S. J. A. S. C. Tesfamariam, "Constrained multi-objective optimization algorithms: Review and comparison with application in reinforced concrete structures," vol. 83, p. 105631, 2019.
- [57] X. Zhang, X. Zheng, R. Cheng, J. Qiu, and Y. J. I. S. Jin, "A competitive mechanism based multi-objective particle swarm optimizer with fast convergence," vol. 427, pp. 63-76, 2018.
- [58] A. Assad and K. J. I. S. Deep, "A hybrid harmony search and simulated annealing algorithm for continuous optimization," vol. 450, pp. 246-266, 2018.
- [59] W. Sheng, Y. Liu, X. Meng, T. J. C. Zhang, and M. w. Applications, "An Improved Strength Pareto Evolutionary Algorithm 2 with application to the optimization of distributed generations," vol. 64, no. 5, pp. 944-955, 2012.
- [60] Z. Cui, Y. Chang, J. Zhang, X. Cai, W. J. S. Zhang, and E. Computation, "Improved NSGA-III with selection-and-elimination operator," vol. 49, pp. 23-33, 2019.
- [61] M. Hamdy, A.-T. Nguyen, J. L. J. E. Hensen, and Buildings, "A performance comparison of multi-objective optimization algorithms for solving nearly-zero-energy-building design problems," vol. 121, pp. 57-71, 2016.
- [62] J. Luo *et al.*, "A decomposition-based multi-objective evolutionary algorithm with quality indicator," vol. 39, pp. 339-355, 2018.
- [63] J. Rada-Vilela, M. Chica, Ó. Cerdón, and S. J. A. S. C. Damas, "A comparative study of multi-objective ant colony optimization algorithms for the time and space assembly line balancing problem," vol. 13, no. 11, pp. 4370-4382, 2013.
- [64] S. Selvakumar and R. J. M. T. P. Ravikumar, "A novel approach for optimization to verify RSM model by using multi-objective genetic algorithm (MOGA)," vol. 5, no. 5, pp. 11386-11394, 2018.
- [65] H. Seada and K. J. C. r. Deb, "U-NSGA-III: A unified evolutionary algorithm for single, multiple, and many-objective optimization," vol. 2014022, 2014.
- [66] Z. Liao, Z. Wu, Y. Li, Y. Zhang, X. Fan, and J. J. S. C. Wu, "Core-reviewer recommendation based on Pull Request topic model and collaborator social network," vol. 24, no. 8, pp. 5683-5693, 2020.

- [67] M. Fejzer, P. Przymus, and K. J. J. o. I. I. S. Stencel, "Profile based recommendation of code reviewers," vol. 50, no. 3, pp. 597-619, 2018.
- [68] J. Jiang, J.-H. He, X.-Y. J. J. o. C. S. Chen, and Technology, "Coredevrec: Automatic core member recommendation for contribution evaluation," vol. 30, no. 5, pp. 998-1016, 2015.
- [69] C. Yang *et al.*, "RevRec: A two-layer reviewer recommendation algorithm in pull-based development model," vol. 25, no. 5, pp. 1129-1143, 2018.
- [70] M. T. Emmerich and A. H. J. N. c. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," vol. 17, no. 3, pp. 585-609, 2018.
- [71] J. Liang, Q. Guo, C. Yue, B. Qu, and K. Yu, "A self-organizing multi-objective particle swarm optimization algorithm for multimodal multi-objective problems," in *International Conference on Swarm Intelligence*, 2018, pp. 550-560: Springer.
- [72] S. Asthana *et al.*, "WhoDo: automating reviewer suggestions at scale," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 937-945.
- [73] A. Strand, M. Gunnarson, R. Britto, and M. Usman, "Using a context-aware approach to recommend code reviewers: findings from an industrial case study," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, 2020, pp. 1-10.
- [74] K. A. Tecimer, E. Tüzün, H. Dibeklioglu, and H. Erdogmus, "Detection and Elimination of Systematic Labeling Bias in Code Reviewer Recommendation Systems," in *Evaluation and Assessment in Software Engineering*, 2021, pp. 181-190.
- [75] C. Hannebauer, M. Patalas, S. Stünkel, and V. Gruhn, "Automatically recommending code reviewers based on their expertise: An empirical comparison," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 99-110.
- [76] M. M. Rahman, C. K. Roy, J. Redl, and J. A. Collins, "Correct: Code reviewer recommendation at github for vendasta technologies," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 792-797.
- [77] M. Chouchen, A. Ouni, M. W. Mkaouer, R. G. Kula, and K. Inoue, "Recommending peer reviewers in modern code review: a multi-objective search-based approach," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 307-308.

- [78] Y. Yuan, H. Xu, and B. Wang, "An improved NSGA-III procedure for evolutionary many-objective optimization," in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, 2014, pp. 661-668.
- [79] B. Li, J. Li, K. Tang, and X. J. A. C. S. Yao, "Many-objective evolutionary algorithms: A survey," vol. 48, no. 1, pp. 1-35, 2015.
- [80] S. Bhutada, V. Balaram, V. V. J. J. o. I. Bulusu, and O. Sciences, "Semantic latent dirichlet allocation for automatic topic extraction," vol. 37, no. 3, pp. 449-469, 2016.
- [81] S. O. J. C. E. Sada, "The use of multi-objective genetic algorithm (MOGA) in optimizing and predicting weld quality," vol. 7, no. 1, p. 1741310, 2020.
- [82] K. Deb, A. Pratap, S. Agarwal, and T. J. I. t. o. e. c. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," vol. 6, no. 2, pp. 182-197, 2002.
- [83] K. Li, R. Chen, D. Savić, and X. J. I. T. o. F. S. Yao, "Interactive decomposition multiobjective optimization via progressively learned value functions," vol. 27, no. 5, pp. 849-860, 2018.
- [84] J. Blank and K. J. I. A. Deb, "Pymoo: Multi-objective optimization in python," vol. 8, pp. 89497-89509, 2020.