Information Retrieval-Based Optimization Approaches for Requirement Traceability Recovery

Danissa Victoria Rodriguez Caraballo

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

Part of the Software Engineering Commons

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_dissertations/5242

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
INFORMATION RETRIEVAL-BASED OPTIMIZATION APPROACHES FOR REQUIREMENT TRACEABILITY RECOVERY

A Thesis

Submitted to the Graduate Faculty of the Louisiana State University and Agricultural and Mechanical College in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in

The Division of Computer Science and Engineering

by

Danissa Victoria Rodriguez Caraballo
B.S. Universidad de Oriente Venezuela, 2010
M.Sc., Louisiana State University, 2016
August 2020
To God, my family, and my country VENEZUELA.
ACKNOWLEDGEMENTS

I would like to thank my adviser Dr. Doris Carver for her guidance and encouragement in every aspect of my education at LSU. Her patience, kindness, and interest toward me have been an invaluable help in the culmination of my Doctoral Degree. Also, I would like to express my gratitude to the members of my exam committee Dr. Anas Mahmoud who has motivated me and helped me in every phase of this work, Dr. Jianhua Chen for her help in reviewing and evaluating my research and Dr. Yuri Antipov for his help in reviewing my work. Additionally, I would like to thank all the professors who became an important part of my education and professional development; thanks to all of those who inspired me and guided during this process.

I would like to thank my parents Sabas Rodriguez and Danis Caraballo for their unconditional support and love. Also, to my brother Jose Daniel Rodriguez and my sister Dafnae Rodriguez who motivated me during hard times and became an example of hard working and determination. To my niece Nathalia and nephews Alejandro and Leonel for their love and admiration that motivate me to be the best version of myself. Thank you for being the best family I could have asked for. Even with the distance, my heart can feel you close to me. Thanks to my aunts, uncles and cousins, especially to my family in the US: uncle Freddy, aunt Rosemary, and cousins Ariel and Christian. Words cannot express how grateful I am for having you around, and for the love and support you have given me. Finally, in memory of my uncle Nelson Caraballo.

Thanks to my loved one Santiago D. Salas for being by my side, giving me the strength and the motivation to keep moving forward. I am thankful for my friends who have become a family away from home and a great support during hard times. Thank you for making this experience a life memory. I greatly value their friendship.
My special acknowledge to the Fulbright Commission for honoring me with the scholarship that made this dream possible, and the Louisiana State University’s Department of Computer Science for opening doors to me.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................................................................................................................. iii

ABSTRACT .......................................................................................................................................................... x

1. INTRODUCTION ........................................................................................................................................ 1
   1.1. Background ........................................................................................................................................ 1
   1.2. Dissertation Motivation ....................................................................................................................... 3
   1.3. Dissertation Organization ....................................................................................................................... 7
   1.4. Research Questions ............................................................................................................................... 8
   1.5. Contributions of Dissertation ............................................................................................................... 9

2. COMPARISON OF INFORMATION RETRIEVAL TECHNIQUES FOR TRACEABILITY LINK RECOVERY .................................................................................................................. 10
   2.1. Introduction ....................................................................................................................................... 10
   2.2. Related work ...................................................................................................................................... 13
   2.3. Methodologies for RTR based in IR .................................................................................................. 18
   2.4. Implementation ............................................................................................................................... 25
   2.5. Experimental results and discussion ................................................................................................. 28
   2.6. Conclusion ....................................................................................................................................... 31

3. MULTI-OBJECTIVE IR-BASED NSGA-II OPTIMIZATION APPROACH FOR REQUIREMENTS TRACEABILITY RECOVERY ......................................................................................... 33
   3.1. Introduction ..................................................................................................................................... 33
   3.2. Background ...................................................................................................................................... 36
   3.3. Related Work ................................................................................................................................... 40
   3.4. Adaptation of the IR-based NSGA-II to solve the RTR problem ......................................................... 45
   3.5. NGSA-II implementation and experimental setup ............................................................................... 53
   3.6. Experimental results ......................................................................................................................... 57
   3.7. Threats to validity ............................................................................................................................. 68
   3.8. Conclusions ................................................................................................................................... 69

4. AN IR-BASED ARTIFICIAL BEE COLONY APPROACH FOR TRACEABILITY LINK RECOVERY ................................................................................................................................. 72
   4.1. Introduction ..................................................................................................................................... 73
   4.2. Background ...................................................................................................................................... 76
## 4. Related work

- **ABC algorithm**
- **Modified ABC to solve the RTR problem**
- **ABC implementation and experiment setup**
- **Experimental results**
- **Threats to validity**
- **Conclusions**

## 5. COMPARISON OF NSGA-II AND ABC ADAPTED TO THE REQUIREMENT TRACEABILITY LINK RECOVERY PROBLEM

- **Introduction**
- **Related work**
- **Computational intelligence adapted to the RTR**
- **IR-based NSGA-II algorithm**
- **IR-based ABC algorithm**
- **Experimental results**
- **Conclusion**

## 6. CONCLUSIONS AND PERSPECTIVES

- **Conclusions**
- **Contributions**
- **Future perspective**

## APPENDIX

- **COPYRIGHT INFORMATION**
- **INSTITUTIONAL REVIEW BOARD APPROVAL**
- **REFERENCES**
- **VITA**
LIST OF TABLES

2.1. Values of P and R over EBT data set ................................................................. 30
2.2. Values of P and R over eTour data set ............................................................... 30
3.1. Representation of an NSGA-II individual ............................................................ 49
3.2. P and R for 10 iterations over EBT data set ....................................................... 58
3.3. P and R for 10 iterations over Albergate data set ............................................. 60
3.4. P and R for 10 iterations over eTour data set .................................................... 61
4.1. Representation of food source ............................................................................. 93
4.2. P and R for 10 iterations over EBT data set ....................................................... 101
4.3. P and R for 10 iterations over Albergate data set ............................................. 103
4.4. P and R for 10 iterations over eTour data set .................................................... 104
4.5. Mean P and R for 10 iterations over EBT, Albergate and eTour data sets .......... 106
5.1. P and R for NSGA-II and ABC for 10 iterations over EBT data set ................. 129
5.2. P and R for NSGA-II and ABC for 10 iterations over Albergate data set .......... 130
5.3. P and R for NSGA-II and ABC for 10 iterations over eTour data set ............... 132
5.4. ABC time complexity ......................................................................................... 136
5.5. NSGA-II and ABC algorithms time complexity ............................................... 137
5.6. NSGA-II and ABC execution time in seconds ................................................. 139
LIST OF FIGURES

2.1. Methodology for the preprocessing phase for traceability link recovery using IR techniques. .................................................................................................................. 19

2.2. IR techniques implementation. ................................................................................................................................. 26

2.3. P and R over EBT data set ......................................................................................................................................... 30

2.4. P and R over eTour data set .................................................................................................................................... 31

3.1. Terminology mapping of RTR to NGSA-II algorithm. ............................................................................................... 47

3.2. Results of application of mutation operator. ............................................................................................................. 50

3.3. Results of application of crossover operator. ........................................................................................................... 51

3.4. Preprocessing implementation .................................................................................................................................. 54

3.5. NSGA-II implementation ............................................................................................................................................. 55

3.6. NSGA-II output for EBT data set .............................................................................................................................. 56

3.7. Values of P and R for 10 iterations over EBT data set ............................................................................................. 59

3.8. Values of P and R for 10 iterations over Albergate data set ..................................................................................... 61

3.9. Values of P and R for 10 iterations over eTour data set ........................................................................................... 62

3.10. Mean values of P and R for 10 iterations over EBT, Albergate and eTour data sets. .............................................. 63

3.11. Comparison of the P and R values of the NSGA-II algorithm and IR techniques over EBT and eTour. ......................... 65

3.12. F1 values of the NSGA-II algorithm and IR techniques over EBT and eTour. .......................................................... 66

3.13. Comparison of P and R of our approach and previous approaches. ................................................................. 68

4.1. Computer intelligence paradigm ............................................................................................................................... 77

4.2. ABC algorithm process ........................................................................................................................................... 86

4.3. Terminology mapping of RTR to ABC algorithm .................................................................................................. 91

4.4. Mutate operation example ......................................................................................................................................... 96
4.5. Preprocessing phase. .......................................................................................................................................................... 96
4.6. ABC algorithm adapted to the RTR problem. ....................................................................................................................... 97
4.7. ABC output. ........................................................................................................................................................................ 99
4.8. Values of P and R for 10 iterations over the EBT data set. ................................................................................................. 102
4.9. Values of P and R for 10 iterations over the Albergate data set. .......................................................................................... 103
4.10. Values of P and R for 10 iterations over the eTour data set. ............................................................................................... 105
4.11. Mean of P and R for 10 iterations over EBT, Albergate and eTour data sets. ............................................................. 106
4.12. Comparison of P and R values of ABC and IR techniques. ................................................................................................. 108
4.13. F1 values of ABC and IR techniques. .............................................................................................................................. 109
4.14. Comparison of P and R values of our ABC approach and genetic algorithms ............................................................... 110
5.1. Comparison of results for the NSGA-II and ABC algorithms over EBT. .............................................................. 130
5.2. Comparison of results for the NSGA-II and ABC algorithms over Albergate. ................................................................. 131
5.3. Comparison of results for the NSGA-II and ABC algorithms over eTour. ................................................................. 133
5.4. F1 results for the NSGA-II and ABC algorithms over EBT, Albergate and eTour. ............................................................... 134
ABSTRACT

Requirements traceability provides support for important software engineering activities. Requirements traceability recovery (RTR) is becoming increasingly important due to the numerous benefits to the overall quality of software. Improving the RTR problem has become an active topic of research for software engineers; researchers have proposed a number of approaches for improving and automating RTR across the requirements and the source code of the system. Textual analysis and Information Retrieval (IR) techniques have been applied to the RTR problem for many years; however, most of the existing IR-based methodologies applied to the RTR problem are semiautomatic or time-consuming, even though many links are correctly recovered using IR. Thus, there is a need for effective and innovative approaches for automatization in the RTR problem. In this research, we study IR techniques applied to the RTR problem to determine the optimal alternative to RTR across the textual content of requirements and system source code, and propose innovative methodologies based on computational intelligence combine with IR to achieve automatization. We approach the study of the RTR problem as an optimization problem; the problem is formulated as a multi or mono objective search in which we assign one-to-many relationships between each requirement and source code classes by considering similarity in their textual content. The Non-dominated Sorting Genetic Algorithm (NSGA-II) and Artificial Bee Colony (ABC), when combined with IR techniques, appear to provide promising alternatives for finding a complete and accurate list of traceability links. We adapt the NGSA-II and ABC algorithms to solve the RTR problem, generate programming tools for experimentation, and report the results on three open source projects. Results show values of precision and recall above 70%. NSGA-II and ABC are also analyzed based on time complexity using the big-O notation; results indicate NSGA-II is more time efficient and less precise than ABC.
1. INTRODUCTION

1.1. Background

Requirements are crucial in software development because they express the functionality of systems. Requirements go through a long process that starts with the requirement engineering phase, followed by specification, design, development, test and deployment. Factors such as size of the application, the number of people participating in the process, rapid evolution of the system, and changes in the original requirements document can cause engineers to be unable to track the requirements through the later phases of the software development process. Lack of requirements tracking can cause problems in the deployed software since there is no way to verify the correct implementation of all requirements. In addition, the system becomes difficult for future modifications.

Requirements traceability, which is defined by Gotel & Finkelstein (1994) as “the ability to describe and follow the life of a requirement, in both a forwards and backwards direction”, is a primary way to show that the source code of a system is consistent with its requirements, and to ensure that developers implemented all the requirements specified at the initial phases of the software development process (Ali et al., 2012). Requirements traceability is an important practice in any software assessment, but it is a key factor when building safety critical and high-assurance software systems in which missing requirements can cause catastrophic consequences (Kelly, 1999).

Traceability is unpopular among stakeholders because of a lack of understanding of its practical benefits (Rochimah et al., 2007). As a result, stakeholders typically do not enforce or request traceability in their systems. However, empirical evidence indicates that traceability
reduces the projected defect rate during software development (Rempel & Mäder, 2016). The expected benefits include effort reduction and output accuracy. Traceable software requirements are critical especially in complex software deployment. The complexity of software systems and the interconnections between requirements demand the use of traceability models and tools for enhancing software fault detection and overall quality (Salem, 2006).

Requirements traceability is important for systems with frequent and continuous changes, especially adaptive systems and systems that exhibit emergent behavior. In systems of this nature, the requirement changes need to be managed and their impact analyzed in detail (Ghannem et al., 2017). For example, when high-level requirements change, lower-level objects usually need to be modified, justifying the utilization of requirements traceability (Spanoudakis et al., 2004).

During software quality assurance and testing, the requirement traceability recovery (RTR) is also beneficial. If a low-level requirement fails during testing, the software engineer can easily identify the high-level requirements that are not executed correctly. Therefore, the affected segments can be identified, documented, reviewed and corrected, based on the requirements traceability, if a malfunction is detected (Salem, 2006). Software evolution also benefits from requirements traceability since it helps to consider important aspects of the software and its changes in order to minimize the transition efforts (Ali et al., 2011; Rochimah et al., 2007).

RTR generates or updates traceability links among the requirements and other design artifacts (Ali et al., 2011). Traceability links provide support for software engineers to gain knowledge about the relations and dependencies across software artifacts resulting from the software development process. Traceability links are the resultant pairs of requirements and other artifacts. Examples include requirements specification and source code which implements the
requirements, design documents and test cases, or requirements and design documents (Tsuchiya, 2013).

RTR holds a significant role in diverse sections of the software life-cycle, and it is broadly identified as a major factor for ensuring software engineering efficiency. In addition, it improves software development and maintenance by supporting numerous activities, including verification, impact of change analysis, program comprehension and software reuse (Ghannem et al., 2017). However, recovering traceability links from systems is a tedious, costly and time-consuming task for software developers. Consequently, the literature provides methods, techniques and tools which aim to efficiently recover traceability links as a semi-automatic or automatic activity (Antoniol et al. 2002).

1.2. Dissertation Motivation

RTR has gained popularity in academia and industry over the past decade. Despite multiple debates on whether or not traceability links are important, many researchers have focused their studies on finding novel techniques which automatically generate traceability links as well as improving accuracy. Nevertheless, most of the proposed tools are time-consuming or semi-automatic. They require user manipulation. Techniques found in the literature to support RTR include Information Retrieval (IR), ontological approach (Zhang et al., 2008) and machine learning approaches (Mahmoud & Williams, 2016), and tools such as REquirements TRacing On-target RETRO (Hayes et al., 2007), ReqSimile (Och Dag et al., 2005), Poirot:TraceMaker (Lin et al., 2006), ReqAnalyst (Lormans & Van Deursen, 2006), TraceViz (Marcus et al., 2005), and ADAMS Re-Trace (De Lucia et al., 2005).
IR techniques have been considered for some time as an acceptable solution for recovering traceability links from artifacts, such as source code and requirement documents, where textual analysis is key to find relationships (Ali et al., 2011; Antonial et al., 2002; Marcus & Maletic, 2003; Hayes et al., 2004; Maletic & Collard, 2009, Mahmoud & Niu, 2014). The most popular IR techniques adapted to the RTR problem are the IR Probabilistic, Vector Space IR Model (VSM), and the Latent Semantic Indexing (LSI) (Hayes et al., 2007; De Lucia et al., 2005; Ali et al., 2011; Antoniol et al., 2002; Rodriguez & Carver, 2019; Mahmoud & Niu, 2014). Initial stages of our research compare the effectiveness of these three IR techniques for the RTR problem. We adapted the IR techniques to the RTR problem, and evaluated their performance. However, research suggests that making RTR automatic requires the generation of traceability links without having an engineer or analyst to repeatedly and manually review all of the requirements (Deb et al., 2002).

The goal of this research is to develop an improved method to assist the automation of RTR. The approach is intended to recover links between requirements and artifacts, such as source code elements, for helping software engineers to accurately manage their systems. Our goal is to help software engineers find potential mismatches in existing links, evaluate linkage completeness in case of existence of links, and generate traceability where requirements to source code links do not exist.

We investigate the RTR process as a search-based optimization problem. Our approach takes a software system artifact and a set of requirements as input, and applies nature-inspired metaheuristic algorithms to generate a set of traceability links (solution) between the elements (classes) of the system and the requirements. We seek the set of links that best satisfies the objective functions which are mathematical representations of the problem that controls the search process in optimization algorithms. We apply IR-based objective functions, which is motivated by
the popularity of the application of IR techniques to the RTR problems and the rich information provided by textual contents of the system.

Nature-inspired metaheuristic algorithms have become powerful and popular in computational intelligence and have many applications. Computational intelligence and metaheuristic algorithms have become increasingly popular in computer science, artificial intelligence, machine learning, engineering design, data mining, image processing, and data-intensive applications. Nature-inspired metaheuristic algorithms are presently incorporated to assist numerous software engineering activities, including RTR. A Metaheuristic Search (MS) approach performs an intelligent search to find, within a large space search, a good approximate solution, in cases where a precise algorithmic method is not available or time-consuming (complex combinatorial problems). The literature describes many MS techniques, such as hill climbing (Mitchell, 1998), simulated annealing (Kirkpatrick et al., 1983), Swarm intelligence (Karaboga, 2005), and genetic algorithms (Goldberg, 1989).

We use the NSGA-II algorithm, an improved version of a classic genetic algorithm proposed by Deb et al. (2002). NSGA-II is a multi-objective algorithm that works with a large population search space and has a fast convergence rate. We introduce a novel approach for automatic RTR using an optimization algorithm controlled by two IR-established objective functions based on textual similarity measures, and which relies on minimal information from the system, needing only requirements and source code textual content. The optimal solution is the set of links that maximizes one objective function without minimizing the other. Links are pairs formed by a requirement and a source code. The value of each objective function for every possible solution (set of links) is calculated based on the sum of the textual similarity measured for each link in the set.
Additionally, we use a modified version of the swarm intelligence algorithm Artificial Bee Colony (ABC). The ABC is an optimization algorithm proposed by Karaboga (2005). It is based on the intelligent behavior of a honeybee swarm. Swarm intelligence (SI) is a sub-field of computational intelligence (CI) focused on developing bioinspired multi-agent intelligence systems. It uses the collective behavior of natural agents, such as bees, to create the algorithms (karaboga, 2005). These algorithms have proven to be efficient for solving real-world problems.

Mathematically speaking, to solve a real-world optimization problem using CI algorithms, a mathematical representation of the problem is needed. This representation is the objective function, which is a mathematical rule that describes the problem and its decision variables (Karaboga et al., 2014; Karaboga & Basturk, 2007).

Our approach, based on ABC, uses a set of requirements and the source code of a software system as the input. It generates a set of links from the requirements to the source code as the output. The output is the set that maximizes the objective function which is based on the application of the Vector Space Model approach and uses a weighted cosine similarity. The weights are based on the frequency $TF-idf$ to generate the vectors. The objective function is defined by the sum of all the percentages of similarities resulting from the evaluation of each link in the set. A higher value for the objective function translates to more matches. The set of links that maximizes the objective function is defined as the optimal solution.

Use of NSGA-II and ABC are investigated, executed, tested and compared to define the optimal solution to the RTR problem and to find a tradeoff among precision (P), recall (R) and time complexity.
1.3. Dissertation Organization

This dissertation is structured in six chapters. The first chapter highlights the motivation and main aims of this work. It includes a description of the RTR problem and the proposed solutions based on computational intelligence.

Chapter 2 presents the most popular IR techniques applied to the RTR problem. It highlights the application of IR probabilistic, Vector Space Models and Latent Semantic Index for finding textual similarities among requirements and other design artifacts. These techniques use the textual content of system artifacts to establish similarities. The application of IR techniques to the RTR problem assumes that a strong similarity among the components of a system suggests a possible link.

Chapter 3 introduces a novel approach based on the application of a nature-inspired metaheuristic algorithms to the RTR problem. It approaches RTR as an optimization problem, and uses a multi-objective genetic algorithm called NSGA-II. The RTR problem is modelled using IR-based objective functions which controls the execution of the NSGA-II genetic algorithm. The approach aims to generate traceability links across requirements and the source code of the system by a combined application of the NSGA-II and IR-techniques.

Chapter 4 presents a novel approach based on the study of the RTR problem as a complex combinational problem. We apply an IR-based Swarm intelligence approach to intelligent search within a large search space (all possible combinations of requirements and source code classes pairs) the list of links that maximizes the objective function based on vector space models. We apply a nature-inspired metaheuristic algorithm which belongs to the swarm intelligence field, the artificial bee colony (ABC).
Chapter 5 compares the two approaches defined in Chapters 3 and 4, respectively, to define a best alternative for the RTR problem. NSGA-II and ABC algorithms are from the computational intelligence field; however, they belong to different subclasses. The two algorithms have different internal logics to solve the same problems, which contributes to an interesting comparison.

Chapter 6 provides conclusions from the application of optimization and metaheuristic search to the RTR problem, and future work in this area.

1.4. Research Questions

We address the following research questions in this dissertation research:

RQ1: Can an intelligent search approach assist RTR?

RQ2: Can the multi-objectives genetic algorithm NSGA-II controlled by IR-based objective functions assist automatic RTR?

RQ3: How effective is the application of a MS approach to the RTR problem in contrast to IR-techniques?

RQ4: Can the SI algorithm ABC and an IR-based objective function assess automatic RTR?

RQ5: How effective is the application of SI algorithm to the RTR problem in contrast to IR techniques?

RQ6: How effective is the application of a SI algorithm to the RTR problem in contrast to genetic algorithms?
1.5. Contributions of Dissertation

The original contributions from this research are:

- A deep analysis of IR techniques applied to the RTR problem. We evaluate and compare the most popular IR techniques used in the recovery of traceability links which are based on the textual content of requirements and source code to establish relationships among them.

- A novel approach that applies an IR-based NSGA-II algorithm to the RTR problem. We present a feasible solution to automatize the RTR problem which relies only on essential information of the system and find traceability links among requirements and source code classes using the textual content of both components with approximately 70% of precision (P) and recall (R).

- A novel approach that applies ABC, a swarm intelligence algorithm, to the RTR problem. We adapted the original ABC algorithm to work with textual data input, and presented a solution that recovers traceability links across requirements and source code classes of the system by establishing textual similarities between the textual content of both components. Our implementation returns the optimal list of traceability links with values above 70% P and R.

- A comparison between two nature-inspired metaheuristic algorithms to evaluate their effectiveness when applied to the RTR problem in terms of the IR metrics of P and R and to determine their efficiency in terms of time complexity. By comparing both algorithms, we determine the better alternative to achieve automatization in the RTR problem.
2. COMPARISON OF INFORMATION RETRIEVAL TECHNIQUES FOR TRACEABILITY LINK RECOVERY*

Requirements traceability is a primary means to address completeness and accuracy of requirements. It is an active research topic for software engineers. Textual analysis and information retrieval techniques have been applied to the requirements traceability recovery problem for many years due to the textual components of requirements and source code. Information retrieval techniques are semiautomatic techniques for recovering traceability links, and, on occasion, they have become the baseline for automatic methods applied to requirements traceability recovery. We evaluate the performance of IR techniques applied to the requirement traceability recovery process.

The most popular information retrieval techniques applied to the requirements traceability recovery problem are the IR Probabilistic, Vector Space Model, and Latent Semantic Index techniques. All three techniques rank documents by using one of the documents for extracting queries and the other as the documents being searched using those extracted queries; however, they apply different internal logics for establishing similarities. We compare IR Probabilistic, Vector Space Model, and Latent Semantic Index approaches to evaluate their performance for requirement traceability recovery using the metrics of precision and recall. Experimental results indicate a low precision and recall for the LSI technique and high precision and low recall for both the IR probabilistic and the VSM techniques.

2.1. Introduction

The definition and understanding of the requirements are the first step of every software system which starts with the requirement engineering phase and culminates with the deployment

* This Chapter was previously published as © [2019] IEEE. Reprinted, with permission, from [Rodriguez, D. V., & Carver, D. L., Comparison of Information Retrieval Techniques for Traceability Link Recovery, IEEE 2nd International Conference on Information and Computer Technologies (ICICT) (pp. 186-193), 2019, March].
phase. Requirements are often added later in the development process. During the later phases of the software development process, software engineers often have difficulty tracking requirements, due to factors such as size of the application, the number of people participating in the process, rapid evolution of the system, and changes in original requirements documents. Lack of keeping requirements traceability links updated introduces errors in the final software product since it is difficult to corroborate the correct implementation of all requirements and verify completeness.

Requirements traceability, which traces the life of a requirement from its creation to implementation (Gotel & Finkelstein, 1994), becomes indispensable for the development of safety critical software systems in which missing requirements can cause catastrophic consequences (Kelly, 1999). Empirical evidence indicates that traceability reduces the expected defect rate during software development (Rempel & Mäder, 2016). The complexity of software systems and the interconnections between requirements demand the use of traceability models and tools for enhancing software fault detection and overall quality (Salem, 2006). Regardless of the benefits of requirements traceability, traceability is not always popular among stakeholders (Rochimah et al., 2007). As a result, stakeholders regularly do not enforce or request traceability in their systems, often because of time and cost.

Requirements Traceability Recovery (RTR) is a process for generating or updating traceability links among the requirements and other design artifacts (Salem, 2006). A good implementation of RTR requires that misunderstandings and ambiguities are eliminated from requirements documents (Rodriguez et al., 2018; Mahmoud & Carver, 2015), so that requirements documents provide meaningful information for RTR to assist better software development and maintenance.
Despite differing opinions about the importance of traceability links, researchers are working to find novel techniques for automatic generation of traceability links, focusing not only on automatization but also on accuracy. Information Retrieval (IR) techniques are a popular and a broadly applied solution for recovering traceability links from artifacts, such as source code and requirement documents. Textual analysis is key for finding relationships (Ali et al., 2011; Antoniol et al., 2002; Maletic & Collard, 2009). IR techniques find relevant documents among document groups using queries; techniques for retrieval based on keywords are the most studied and broadly used (Hayes et al., 2003). In IR techniques based on keywords search, each document in one document set is analyzed to extract the keywords or important terms to find in the second document. These keywords are compared against the words associated within each document to the other document set to determine document similarity (Hayes et al., 2003). Many of the keyword-based techniques use a similarity measure that computes the relatedness between the representations of a document and a query.

The most popular IR techniques adapted to the RTR problem are the IR Probabilistic, Vector Space IR Model (VSM), and the Latent Semantic Indexing (LSI) (Ali et al., 2011; Antoniol et al., 2002; Maletic & Collard, 2009). The goal of this research is to compare the effectiveness of these three IR techniques for the RTR problem. Section 2.2 discusses related work, section 2.3 describes the methodologies for RTR based on IR, section 2.4 explains the implementation of the IR methodologies adapted to the RTR process, and section 2.5 contains the experimental results and discussion of the results. Section 2.6 presents conclusions and future work.
2.2. Related work

We investigate and compare the application of different IR techniques to the RTR problem. This approach requires exploring the conceptual bases of traceability link recovery and the most popular IR techniques which could be adapted to the RTR problem.

2.2.1. Traceability Link Recovery

The recovery of traceability links across software artifacts is a challenging task. During the development of software systems, there should be a corresponding evolution of the traceability links across different software artifacts, but developers often do not keep links updated. As a result, the failure to maintain traceability links increases the difficulty of identifying relationships across different types of artifacts, affecting program comprehension, maintenance, requirements tracing, impact analysis and reuse.

Traceability links established across textual documentation and source code facilitate several tasks. Traceability link recovery facilitates program comprehension, because it supports both top-down and bottom-up comprehension by creating links between parts of code and associated sections of textual documents (Cleland-Huang et al., 2005). Maintenance of software benefits by traceability link recovery. Links between code and other artifacts such as design documentation facilitate the utilization of general knowledge related to the problem and application domains for performing analysis, thereby reducing the complexity of associating domain concepts with fragments of the code.

Traceability link recovery is important for requirement tracing; links are necessary to identify the areas of code involved in the implementation of a specific requirement (Soloway & Ehrlich, 1984), evaluate the completeness of an implementation according to the requirements
document, and formulate complete and comprehensive test cases. Additionally, traceability link recovery is used for studying the impact on a product affected by a planned change and for assisting in reuse of existing software, since links across code to textual documents help to locate possible reusable components (Von Mayrhauser & Vans, 1993).

2.2.2. Information Retrieval

The software engineering community, both research and commercial, have combined efforts to improve the correlation of documentation, such as a requirement specification document, and source code (Oliveto et al., 2010). Encouraging results have been found by the application of IR techniques for traceability recovery (Spanoudakis et al., 2004). Several researchers apply IR techniques (De Lucia et al., 2008) to semi-automatically recovered traceability links across artifacts of different types (Antoniol et al., 2002; Maletic & Collard, 2009). Such techniques recover traceability links based on the similarity between the textual content in the software artifacts. Techniques are based on software documentation being expressed in textual form, and in the assumption that programmers use meaningful terms to name the source code identifiers (De Lucia et al., 2008). Techniques based on IR generate a list of possible traceability links used as a reference to the similarity between textual content of different software artifacts. The fundamental idea is the assumption that two artifacts with high textual similarity are related and they should be traced to each other. In our implementation, we use source code as a query and the requirements specification document as the document being searched.

We measure the effectiveness of IR techniques using the IR metrics of recall and precision (Antoniol et al., 2002; Mahmoud & Carver, 2015). Recall can be defined as the percentage of traceability links recovered correctly within the total number of links. Precision is defined as the percentage of traceability links correctly recovered from the total amount of links recovered. In
some cases, there can be high recall value, but the precision value is zero or vice versa. These cases are not desirable since in both cases developers need to perform a manual revision of the results to remove false positives and analyze the source code in order to recover missing links (Antoniol et al., 2002). There needs to be a balance between precision and recall.

IR techniques are good options for semi-automatic recover of traceability links between the software design textual documents and its source code. Similar to (Biggerstaff et al., 1993) and (Antoniol et al., 2002), a foundation of our work is that source codes are written using a good programming style, and developers use meaningful names for identifiers (methods, variables, and classes). In some implementations of IR techniques, such as the techniques described in Biggerstaff et al. (1993), the names of identifiers in the code hold ideas to suggest concepts implemented in the code. However, we follow the idea expressed by Antoniol et al. (2002) that the names of identifiers are the key to finding similarities by finding matching parts of the documentation. Regardless of the IR technique utilized to analyze the text and find similarities, a key constraint is that both documents participating in the process must have a common representation given by their terms and context. Documents should be written in the same language. In the case of source code, we assume that developers maintained the same natural language in documentation as well as name of identifiers, and we assume that meaningful names were used in the code.

IR techniques are fundamental for many semi-automatic tools such as algebraic or probabilistic models (Antoniol et al., 2002), data mining (Zhang et al., 2008), machine learning (Spanoudakis et al., 2004), cross-referencing schemes (Bratman & Court, 1975), scenarios (Bouillon et al., 2013), and key phrase dependencies (Jackson, 1991). An in-depth survey can be found in (Gotel & Finkelstein, 1994). Many tools for RTR are found in the literature, including
Retro (Requirement Tracing On-target) (Hayes et al., 2007), ADAMS Re-Trace (De Lucia et al., 2005), and Trustrace (Ali et al., 2012). Trustrace combines IR and data mining techniques to generate traceability links between requirements and source code (Ali et al., 2012).

2.2.3. Ranked Retrieval Approach

A ranked retrieval approach is “an approach customized to work with different IR models” (Frankes & Baeza-Yates, 1992). It ranks the textual documents from requirements documents against queries extracted from the identifiers of source code components.

2.2.3.1. IR probabilistic

In the probabilistic model, textual documents are ranked utilizing a statistical approach, which is based on the probability of a document being relevant to a query. The ranking exploits the idea of a language model, which is a probability distribution over a sequence of words, that is, “a stochastic model that assigns a probability to every string of words taken from a prescribed vocabulary” (Cullum & Willoughby, 2002). For each document, a language model is estimated, which is constructed by a unigram approximation of the model; a unigram is a contiguous sequence of n items when n=1 from a given sample of text or speech. A Bayesian classifier is used to calculate the total value similarity of the queries extracted from each source code component in comparison to the requirements. A high value is interpreted as a signal of the existence of a traceability link due to a semantic correlation across the elements.

2.2.3.2. VSM and LSI

VSM represents textual content of requirements documents and queries extracted for the source code as vectors in an n-dimensional space, where n is the number of words in the vocabulary. In this research, documents are compared against queries and ranked by computing a
distance function based on the cosine of the angle across the corresponding vectors (Hayes et al., 2007; Rich & Waters, 1988).

VSM does not consider semantic relations between terms. For instance, if a document has the word “automobile” and another document has the word “car”, VSM does not find the match, and it does not establish any similarity measure across the document. As a result, VSM generates a problem in the retrieval of relevant documents, known as the synonym problem. LSI attempts to solve the synonym problem by finding similarity across documents that are related, but that do not share terms.

LSI subspace is an improvement of the VSM that captures the most significant factors of the term-by-document matrix, producing efficiency in finding the relations of most frequent co-occurring terms. LSI “uses singular-value decomposition (SVD) factors to convert a term-by-document matrix into the largest one-dimensional projections of the document vectors” (Antoniol et al., 2002). In LSI, each of the document vectors can be observed as a linear combination of terms. In that sense, “LSI can be referred to as a corpus-based statistical method” (Antoniol et al., 2002).

VSM does not use word order, syntactic relations, or morphology analysis when it is applied to natural language text. However, despite the simplicity of VSM, it produces a good vector representation and results (Ali et al., 2011). VSM works better than LSI for the RTR problem of finding similarities across textual content of design documents and source code because important information is provided by the names of individual words. The position of these words for assigning context to the word loses importance (Maletic & Collard, 2009).
2.3. **Methodologies for RTR based in IR**

Our methodology for traceability link recovery across source code and textual documentation has a preprocessing phase and an implementation phase. We use as a query the identifiers (words) obtained from a source code component to retrieve the requirements related to the component. We analyze identifiers, and we split a sequence of words into single identifiers to keep single “words” for extracting real semantic meaning. The steps of the methodology are: (i) preprocessing to remove stop words and to perform stemming, (ii) analyzing input documents such as the requirements document and source code to construct a representation of each document, and (iii) determining similarities by using different ranking algorithms.

### 2.3.1. Preprocessing

We use the source code to create queries, and those queries are compared against the requirements document. Figure 2.1, which illustrates the preprocessing, shows two paths: the document path that prepares the document for retrieval and the code path that obtains the queries from the source code. The requirement document path generates a vocabulary from the documents and indexes documents according to the vocabulary. The document path transforms capital letters into lower case letters, removes stop-words, transforms plurals into singulars, and performs morphological analysis as described in (Antoniol et al., 2002).
The indexing and ranking of the documents against a query vary depending on the IR technique. Preprocessing may be adjusted depending on the artifacts used for generating traceability links.

2.3.2. Probabilistic IR model

The probabilistic IR model “calculates the ranking scores as the probability that a document $D_i$ (requirement in the requirements document) has similarity to the source code component $SC$ (query $SC$)” (Antoniol et al., 2002). To determine if a document $D_i$ has a relation to the source code component $SC$, a similarity Eq. (2.1) is used.

$$\text{Similarity}(D_i, SC) = P_r(D_i|SC)$$

(2.1)

Applying to Bayes rule (Tucker, 2014), the conditioned probability in Eq. (2.1) can be transformed to Eq. (2.2):

$$\text{Similarity } (D_i|SC) = \frac{P_r(SC|D_i)P_r(D_i)}{P_r(SC)}$$

(2.2)
If we assume that all code documents have equal probability, then, for a given SC, all documents $D_i$ are ranked by the conditioned probabilities $P_r(SC|D_i)$. These probabilities are calculated by estimating a stochastic language model (Mori, 1997) for each document $D_i$. If we assume that the source code components and the requirements document share the same vocabulary $V$, SC can be expressed as a sequence of $m$ words $W_1; W_2;...; W_m$, where $W_i$ are the identifiers of the source code component of the vocabulary $V$. The conditioned probability is defined in Antoniol et al. (2002) as:

$$P_r(SC|D_i) = P_r(W_1, W_2, ..., W_m|D_i) \quad (2.3)$$

The probability occurring in Eq. (2.3) can be expressed according to Antoniol et al. (2002) as:

$$P_r(W_1, W_2, ..., W_m|D_i) = P_r(W_1|D_i) \prod_{k=2}^{m} P_r(W_k|W_1, ..., W_{k-1}, D_i) \quad (2.4)$$

When $m$ has a high value, the probability equation becomes difficult to estimate. Equation (2.4) is simplified by conditioning the dependence of each word to the last $n - 1$ words, where $n < m$, using Eq. (2.5):

$$P_r(W_1, W_2, ..., W_m|D_i) \approx P_r(W_1, W_2, ..., W_m|D_i) \prod_{k=n}^{m} P_r(W_k, ..., W_{k-n+1}, ..., W_{k-1}, D_i) \quad (2.5)$$

The n-gram approximation that assumes a time invariant Markov process (Cover & Thomas, 2012) reduces the data needed to compute $P_r(SC|D_i)$. However, n-gram models are difficult to estimate since, if $|V|$ is the size of the vocabulary, we need to consider all possible $|V|_n$ sequences of words in the vocabulary. Even for a 2-gram (bigram) model, the estimation is demanding.
In our approach, we use unigrams \((n = 1)\) which considers all words \(W_k\) to be independent. Each \(D_i\) is represented by a language model where unigram probabilities are estimated for all words in the vocabulary, and the similarity is computed as in Antoniol et al. (2002) using Eq. (2.6):

\[
\text{Similarity}(D_i, SC) = P_r(SC|D_i) = P_r(W_1, W_2, ..., W_m|D_i) = \prod_{k=1}^{m} P_r(W_k|D_i)
\] (2.6)

In our implementation, whenever any word \(W_k\) is not present in the document \(D_i\), the value for the whole expression is zero since we are using the product \(\prod_{k=1}^{m} P_r(W_k|D_i)\) for calculating term frequency. The problem is known as “the zero-frequency problem” (Witten & Bell, 1991). A solution that analyzes the unigram probability distribution by computing the probabilities using Eq. (2.7) is described in (Mori, 1997):

\[
P_r(W_k|D_i) = \begin{cases} \frac{c_k-\beta}{N} + \lambda & \text{if } W_k \text{ occurs in } D_i \\ \frac{\lambda}{N} & \text{otherwise} \end{cases}
\] (2.7)

where \(N\) is the total number of words in document \(D_i\), and \(c_k\) is the number of occurrences of words \(W_k\) in the document \(D_i\). The interpolation term is defined in (Ney & Essen, 1991) by Eq. (2.8):

\[
\lambda = \frac{n}{N+|V|} \beta,
\] (2.8)

where \(n\) is the total of different words of the vocabulary \(V\) present in document \(D_i\). The value of parameter \(\beta\) is computed according to Ney & Essen (1991) by using Eq. (2.9):

\[
\beta = \frac{n(1)}{n(1)+2\cdot n(2)}
\] (2.9)

where \(n(j)\) is the number of words occurring \(j\) times in document \(D_i\).
2.3.3. Vector space IR model

VSM is a method broadly used for representing documents as vectors (Chowdhury, 2010). “It encodes a document collection by a term-by-document matrix where \([i, j]\)th element indicates the association between the \(i_{th}\) term and \(j_{th}\) document” (Chowdhury, 2010). VSM space is “a geometric representation of a term-by-document matrix as a set of document vectors occupying a vector space spanned by terms” (Chowdhury, 2010). It represents one type of text unit (documents) by its association with the other type of text unit (terms), measuring the association based on term occurrences in the documents. “The similarity between documents is typically measured by the cosine or inner product between the corresponding vectors, which increases as more terms are shared” (Marcus & Maletic, 2003). Two documents are considered similar if their corresponding vectors in the VSM space point have the same general direction (Marcus & Maletic, 2003).

Vector space IR models “map each document and each query onto a vector” (Marcus & Maletic, 2003). For our implementation, in each position of the vector there is a word which is part of the vocabulary extracted from the documents. If \(|V|\) is the size of the vocabulary, then the vector \([d_{i,1}; d_{i,2}; ... d_{i,j}; |V|]\) represents document \(D_i\). The \(j_{th}\) element \(d_{i,j}\) is a degree of the weight of the \(j_{th}\) term of the vocabulary in document \(D_i\). We use different measurements for computing the weights. The simplest way is by using a Boolean variable that will be assigned the value 1 if the \(j_{th}\) term occurs in document \(D_i\), or 0 in case of zero occurrences. The use of the Boolean variable cannot be applied to all scenarios since some cases require complex measures such as metric frequency of the terms in the documents \((TF - idf)\) for calculating the weight (Rich & Waters, 1990).
For the $TF-idf$ metric, “the $j_{th}$ element $d_{i,j}$ is derived from the term frequency $TF_{i,j}$ of the $j_{th}$ term in the document $D_i$ and the inverse document frequency $idf_j$ of the term over the entire set of documents” (Rich & Waters, 1990). The term frequency $TF_{i,j}$ “is the ratio between the numbers of occurrences of word $j_{th}$ over the total number of words contained in the document $D_i$” (Rich & Waters, 1990). The inverse document frequency $idf_j$ is defined in Antoniol et al. (2002) as:

$$idf_j = \frac{\text{Total number of documents}}{\text{Number of documents containing the } j_{th} \text{ term}}$$ (2.10)

The vector element $d_{i,j}$ is defined in (Antoniol et al. 2002) as:

$$d_{i,j} = tf_{i,j} \cdot \log(idf_i)$$ (2.11)

The term $\log(idf_i)$ is a weight for the frequency of a word in a document; the weight, which increases proportionally to the number of occurrences of the word, is specific to the document. The list of identifiers extracted from a source code component SC is represented in a similar way by a vector $[q_1; q_2; \ldots; q_{|V|}]$. Overall the similarity value across $D_i$ and SC is computed as the cosine of the angle between the corresponding vectors defined in Antoniol et al. (2002) as Eq. (2.12):

$$Similarity (D_i, SC) = \frac{\sum_{j=1}^{V} d_{i,j}q_j}{\sqrt{\sum_{i=1}^{V}(d_{i,i})^2 + \sum_{k=1}^{V} q_k^2}}$$ (2.12)

### 2.3.4. Latent semantic indexing

LSI is a “VSM based method for inducing and representing aspects of the meanings of words and passages reflected in their usage” (Deerwester et al., 1990). The experimental results
of applying LSI to natural language text show that LSI performs well on capturing the meaning of single words, and it has good performance on capturing the meaning of sequences of words such as sentences and paragraphs (Dumais, 1991). The fundamental concept of LSI is that word context is important to determine similarity of meaning of a set of words.

Typically, LSI for text analysis utilizes a user-constructed corpus to generate a term-by-document matrix. Then the LSI technique constructs a subspace known as LSI subspace by applying Singular Value Decomposition (SVD) to the term-by-document matrix (Chowdhury, 2010).

The mathematical formulation of LSI establishes that the less frequently occurring term combinations from a given document collection are usually excluded from the LSI subspace. By assuming that less frequently co-occurring terms are less mutually-related, and consequently less sensible, it can be claimed that the LSI technique achieves noise reduction (Chowdhury, 2010).

To calculate SVD, a rectangular matrix $X$ is separated into the product of three matrices ($U$, $V$, and $\Sigma$). The matrix $U$ represents the original row entities as vectors of resulting orthogonal factor values. The matrix $V$ represents the original column entities. The matrix $\Sigma$ contains scaling values so that when the three components are matrix-multiplied, the original matrix is reconstructed ($x = U \Sigma V^T$). The columns of $U$ and $V$ are the left and right singular vectors, respectively, which are the result of the monotonically decreasing values of diagonal elements of $\Sigma$, known as the singular values of the matrix $X$ (Chowdhury, 2010).

To reduce the dimensionality of the solution, coefficients in the diagonal matrix are deleted, starting with the smallest coefficient (Chowdhury, 2010). “The first $k$ columns of the $U$ and $V$
matrices and the first (largest) k singular values of X are used to construct a rank-k approximation to X through $X_k = U_k \sum_k V_k^t$” (Chowdhury, 2010).

In the LSI implementation, the columns of U and V are orthogonal, such that $U^t U = V^t V = I_r$ where r represents the rank of the matrix X. $X_k$, constructed from the k-largest singular triplets of X, is the closest rank-k approximation to X. With regard to LSI, “$X_k$ is the closest k-dimensional approximation to the original term-document space represented by the incidence matrix X” (Chowdhury, 2010). By reducing the dimensionality of the matrix, the noise that affects the result is reduced, thus achieving a better performance.

Empirical studies have investigated the best ways to implement LSI. Several term weighting schemes to represent the input term-by-document matrix are found in (Dumais, 1991). The results were evaluated using the metrics precision and recall on the retrieval tasks of the fixed LSI subspace. Additionally, several studies describe that, depending on the LSI subspace, the performance of LSI presents fluctuations (Deerwester et al., 1990; Dumais, 1991).

LSI subspace representation is helpful in text analysis related tasks since it calculates similarity measures across documents in two different ways, by either calculating the cosine across their corresponding vectors or by their length. Also, LSI assists in information retrieval tasks by mapping queries into the LSI subspace to determine which documents are relevant to the query.

2.4. Implementation

We developed tools to support and partially automate the Probabilistic IR, VSM and LSI algorithms for IR. We use a top-down recursive parser to analyze Java source code. The parse trees are traversed, and each time a class is found, comments, and identifiers of attributes, methods, and method parameters are stored in a support file. We discard comments. The RTR methods are based
on the mnemonics utilized for classes, attributes, methods, and parameters that are compared with the requirements document. Figure 2.2 summarizes the implementation of the IR techniques.

We developed LinkPP (Link PreProcessing) for preprocessing. LinkPP uses Java as the programing language. Java has libraries and packages for preprocessing. The input to LinkPP is a list of requirements and classes from the source code to generate as an output a new file of requirements and a new file of source code classes as a .CSV file. The new files contain the meaningful words for our analysis since we removed stop words, punctuation, capital letters, and keep identifiers from the source code. The input for the implementation of the ranking algorithms is the newly created files.

For the implementation of the ranking algorithm, we developed LinkPIR (Link Probabilistic Information Retrieval), LinkVSM (Link Vector Space Models), and LinkLSI (Link Latent Semantic Index) for the IR probabilistic, VSM, and LSI, respectively. LinkPIR implements the Probabilistic IR technique using Java as the programing language. Java has strong libraries and packages for managing text and performing math calculations. LinkPIR takes as input preprocessed source code and requirement document files and finds similarities between fragments.
of the source code (classes) and requirements in the requirements document by applying the probabilistic IR raking algorithms described in section 2.3.2.

LinkVSM implements vector space models. LinkVSM, which uses Python 3.6, reads as input preprocessed requirement and source code files and defines similarities based on the vector machine algorithm described in section 2.3.3.

LinkLSI implements the LSI technique. LinkLSI uses Python 2.7 as the programming language because this version of Python facilitates the use of libraries for semantical analysis of words. LinkLSI takes as input preprocessed requirements and source code files and establishes similarities between files based on LSI. In contrast to the VSM, the LSI technique bases its similarity on the context of words and not just on finding the exact same words.

LinkPIR, LinkVSM, and LinkLSI utilize as input the output generated by the preprocessing phase, and based on different approaches, they each generate an output that we analyze to determine the preferred technique among the three techniques. LinkPIR, LinkVSM, and LinkLSI find similarities between requirements documents and source code in different ways. Our results, which were generated after applying the three techniques over identical data sets, are described in section 2.5.

LinkPIR, LinkVSM, and LinkLSI have similar internal logic. Each one iterates first over the preprocessed source code file where each line of the file is a class from which identifiers were extracted. For each class those identifiers are verified against the preprocessed requirements file. When a match occurs, the value of the similarity, the ID of the class and the requirement are saved. These values are updated if another match with a higher similarity value is found.
At the end of the execution, the output shows the requirement from the requirements document file that has the highest similarity with a specific class. We have exactly one link per class, or none in case of no match. In cases of classes that are linked with more than one requirement, a threshold would be needed which requires the application of machine learning techniques. This case is beyond the scope of this research.

2.5. Experimental results and discussion

We used two open-source projects, EBT and eTour, to compare the IR techniques. These projects are available at http://coest.org/. EBT (Event Based Traceability) has 40 requirements, 50 Java source code classes, 25 test cases, and trace links from requirements to classes. EBT source code is defined in (Cleland- Huang & Christensen, 2003). The eTour data set, a Tour Guide System, has 58 use cases used to derive the requirements, 116 source code classes, and 308 trace links from requirements to classes. The source code is found in (Poshyvanyk et al., 2011). We selected EBT and eTour because they have been developed over the past 10 years, and because they are a good size for analysis. Additionally they have the structure of the data needed for the analysis. All of them have requirements, source code classes and the list of traceability list from requirements to class which are used to validate our results.

We use precision (P) and recall (R) as the comparison metrics. P and R come originally from the IR field. P is equal to the ratio of the number of correct links detected to the total number of links detected, and R is equal to the proportion of the number of correct links detected to the number of correct links. In general, P denotes correctness and R denotes completeness. Higher P and R rates results are desired since they are closer to the best result. To compute P and R, we extract pair-wise true-positive values (TP), false-positive values (FP), and false negative values
(FN) for each output generated for each technique on a specific data set. TPs are elements that correctly are in the output, FPs are elements that are in the output but should not be, and FNs are elements that should be in the output but are not. P and R are defined by Eq. (2.13) and Eq. (2.14).

\[
P = \frac{TP}{TP+FP} \in [0,1] 
\]

\[
R = \frac{TP}{TP+FN} \in [0,1] 
\]

Table 2.1 shows the values of P and R for the EBT data set. The results show that probabilistic IR and VSM over performed in comparison with LSI. Our implementation of LSI uses semantical information of words. The semantic analysis is based on the positions of words in the sentences for defining context and finding overall meaning; however, since the class data set is formed for words (identifiers) extracted from code, and those words do not form sentences, it is hard to extract overall meaning based on position or context.

Probabilistic IR and VSM show high precision (Probabilistic IR = 95%, VSM = 94%), which means that they are able to classify correctly most of the links based on the similarities of the words from requirements and source code classes. However, the recall values are low (Probabilistic IR = 20%, VSM = 34%) which means that the list of links is not complete. Incompleteness happens because in both cases we are only able to find the pair class-requirement with the highest similarity. For each class in the source code class input file, we determine that the requirement from the requirements input data file with the highest probability is related to the class. As a result, we create only one or no link per class. This process may generate an incomplete list of links.
Table 2.1. Values of P and R over EBT data set.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilistic IR</td>
<td>95%</td>
<td>20%</td>
</tr>
<tr>
<td>VSM</td>
<td>94%</td>
<td>34%</td>
</tr>
<tr>
<td>LSI</td>
<td>21%</td>
<td>7%</td>
</tr>
</tbody>
</table>

Figure 2.3 shows a graphical representation of the results obtained for P and R for each IR algorithm over the EBT data set.

Table 2.2 shows the values of P and R applied to eTour data set. For the eTour data set, which is larger than the EBT data set, we obtained similar results. Probabilistic IR and VSM perform better than LSI. This result is based on the same argument, namely that both data sets have the same structure, and the source code classes files are not formed by sentences.

Table 2.2. Values of P and R over eTour data set.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilistic IR</td>
<td>95%</td>
<td>26%</td>
</tr>
<tr>
<td>VSM</td>
<td>93%</td>
<td>26%</td>
</tr>
<tr>
<td>LSI</td>
<td>4%</td>
<td>1%</td>
</tr>
</tbody>
</table>

The existence of a number of words without any particular structure makes it harder to perform a semantic analysis based on context of words or position in the text. In a similar way,
Probabilistic IR and VSM returned for the eTour data set high precision values (Probabilistic IR = 92%, VSM = 95%), and low recall values (Probabilistic IR = 26%, VSM = 26%). Our implementations are able to find exactly one or no links from a class to a requirement. The number of classes determines the maximum number of links in the data set.

Figure 2.4 shows a graphical representation of the results obtained for P and R for each IR algorithm over the EBT data set.

![Figure 2.4. P and R over eTour data set.](image)

Overall, the analysis suggests that the IR techniques have similar behavior on both data sets. LSI did not perform well on either set. In contrast, the IR Probabilistic and the VSM approaches each had high precision but low recall values.

More work needs to be done on improving values of P and R. We are exploring the combination of IR techniques with other techniques, such as machine learning, that could help automate and increase the recall value while keeping a high precision value.

### 2.6 Conclusion

We applied the most popular IR techniques to the RTR problem. Our methodology was based on two phases, a preprocessing phase that prepares the documents for analysis and the
implementation phase which ranks the documents depending on the IR technique. We studied IR probabilistic, VSM and LSI. To compare and analyze their performance, we implemented each of the three techniques, and applied the three techniques to two data sets. We found higher P and R for IR probabilistic and VSM than LSI, however, R was low for all of the three techniques. These results suggest that more data sets need to be tested and that more research needs to be done to improve the recall values. High precision values suggest that the techniques could be used with other methodologies to improve the RTR process.
3. MULTI-OBJECTIVE IR-BASED NSGA-II OPTIMIZATION APPROACH FOR REQUIREMENTS TRACEABILITY RECOVERY

The recovery of requirements traceability is crucial to the software development process since it facilitates different activities, including analyzing impact of changes, validating requirements, maintenance and evolution of software. A number of solutions for automatic recovery of links across software artifacts have been developed, including the application of textual analysis and information retrieval techniques. Link recovery with information retrieval has seen some success, but there is a need for methods that further enable the automatic generation of links. We investigate the requirements traceability recovery problem as an optimization problem; we study it as a multi-objective search in which we match each requirement to a code element by considering the similarity between their textual content. We adapt the Non-dominated Sorting Genetic Algorithm, NSGA-II, to seek the optimal solution between two objective functions based on textual similarity using the similarity measures of Jaccard similarity and term frequency (TF-idf) combined with cosine similarity. Our IR-based application of NSGA-II returned values on average above 75% for both precision and recall for the recovered links. High precision and recall are positive results for our approach to requirements traceability recovery.

3.1. Introduction

Requirements Traceability Recovery (RTR) is the mechanism that generates or updates traceability links across the requirements and other design artifacts, including source code (Rempel & Mader, 2016; Pennington, 1987; Mahmoud & Carver, 2015). Links to trace code to textual content of requirements document are a practical help to locate possible related components (Caldiera & Basili, 1991); they are crucial to detect the specific segments of the code implementing a particular functional requirement (Pinheiro & Goguen, 1996), to assess the completeness of the
project according to the requirements document, and to generate effective test cases (Arnold & Bohner, 1993). However, traceability is often unpopular among stakeholders because of a lack of understanding of its benefits (Rochimah et al., 2007).

RTR is important for systems with frequent and continuous changes. In such systems, changes to requirements need to be managed and their impact analyzed (Mahmoud & Carver, 2015). RTR also is beneficial during software quality assurance and testing, if a component (class) of the source code fails during testing, the software engineer can identify the requirement or requirements that have not been met. If a malfunction is identified, the affected segments can be identified, documented, reviewed and corrected based on requirements traceability (Ghannem et al., 2017). However, manual generation of these links is a tedious and time consuming activity that is not usually a priority during software development.

Thus, due to the numerous benefits of RTR, researchers have concentrated their studies on finding novel techniques for automatic generation of links. IR techniques generally are considered as good solutions for RTR among software artifacts where textual analysis and similarity in the textual contents of the artifacts are fundamental for establishing relationships. The most frequent IR techniques applied to the RTR problem are IR Probabilistic, Vector Space Model (VSM) and Latent Semantic Indexing (LSI) (Hayes et al., 2007; De Lucia et al., 2005; Ali et al., 2011; Antoniol et al., 2002; Rodriguez & Carver, 2019; Mahmoud & Niu, 2014), but most of the proposed tools are time-consuming or semi-automatic that need user manipulation. However, inspired by the popularity of IR techniques and precision (P) values, our goal is to automate the RTR process. We use IR as the baseline for our approach in which we compare P and recall (R).
Automated tracing is the capacity to create traceability links across software artifacts automatically without requiring the intervention of an operator. The ultimate goal is to obtain 100% accuracy denoted by \( P \) and 100% coverage denoted by \( R \). Even though several attempts to solve the automated tracing problem have been made, techniques that produce results that are more accurate and less dependent on human intervention are still needed (Mahmoud et al., 2012).

We developed a method to assist the automation of RTR which recovers links between requirements and source code elements. We find potential mismatches in existing links, evaluate linkage completeness where links already exist, and generate traceability links where requirements links do not exist. We investigate the RTR process as a search-based optimization problem. Our approach uses source code and requirements as inputs and generates a set of traceability links between the source code classes and the requirements. We search for the set of links that best satisfies the objective functions, which are based on semantic similarity computed using IR techniques.

In this research, we use the NSGA-II algorithm, a stronger version of a classic genetic algorithm proposed by Deb et al. (2002). NSGA-II is a multi-objective algorithm that works with a large population search space and has a fast convergence rate. The main contribution is a novel approach for automatic RTR using an optimization algorithm controlled by two objective functions based on textual similarity measures. The approach relies on minimal information of the system, needing only requirements and source code textual content.

We address the following research questions:

RQ3-1: Can the multi-objective genetic algorithm NSGA-II controlled by IR-based objective functions assist automatic RTR?
RQ3-2: How effective is the application of a MS approach to the RTR problem in contrast to IR techniques?

The remainder of this paper is structured as follows: section 3.2 discusses background information, section 3.3 presents related work, section 3.4 describes the methodologies for the IR based NGSA-II algorithm adapted to traceability link recovery, section 3.5 explains the setup of the experiment, section 3.6 contains the experimental results, section 3.7 explains the treats to validity, and section 3.8 presents conclusions.

3.2. Background

3.2.1. Multi-objective evolutionary algorithm

A multi-objective optimization problem is described as the minimization or maximization of an objective function \( f \) that searches a set of \( m \) parameters (decision variables) that generate a set of \( n \) optimal outputs (objectives) (Zitzler & Thiele, 1999). The optimization problem is described in (Zitzler & Thiele, 1999) as follows:

\[
\begin{align*}
\min / \max & = f(x) = (f_1(x), f_2(x), \ldots, f_n(n)) \\
\text{Subject to} & \\
x & = (x_1, x_2, \ldots, x_m) \in X \\
y & = (y_1, y_2, \ldots, y_m) \in Y
\end{align*}
\] (3.1)

where \( x \) denotes the decision variables vector contained in the solution domain \( X \), and \( y \) represents the objective vector contained in the space \( Y \). The tuples of solutions in multi-objective optimization hold for all decision vectors for which the improvement of the values of the objectives is not feasible in any dimension without deterioration in another dimension (Zitzler & Thiele, 1999). This set of decision vectors is referred to as pareto-optimal.
The decision vectors that do not appear to be dominated by any other decision vector in a given set are referred to as non-dominated vectors. Decision vectors holding the non-dominated characteristic within the entire solution domain are pareto-optimal and constitute the pareto-optimal set or pareto optimal front (Zitzler & Thiele, 1999). The search for a solution in multi-objective problems relies on the evaluation of each of the objective functions. A best solution is determined based on the particular objective by analyzing the values of the objective functions. The ultimate goal in multi-objective optimization is to obtain trade offs between conflicting multiple objective functions (Fleck et al., 2017; Murata et al., 1996; Konak et al., 2006).

3.2.2. Genetic algorithms

Holland developed the concept of genetic algorithms (GAs) in the 1960s (Holland, 1992). GAs are based on the theory of evolution that establishes the origin of a species. The theory establishes that weaker species are more likely to go extinct as a natural selection process, and stronger species are more probable to survive and to pass their genes to the next generations. If a change in the genes of a species is an improvement, the new species evolves and includes the old genes. Changes that do not represent an improvement for the species are discarded by natural selection (Konak et al., 2006; Holland, 1992).

In GA terminology, an individual or chromosome is a solution vector $x \in X$. Chromosomes are formed of single units, and each unit is known as a gene. Usually, a chromosome is associated to a solution $x$ that is unique in the solution space (Konak et al., 2006). A GA operates over a population, which is a collection of chromosomes. It starts by generating a random population then performs an iterative process to search for the optimal solution. During the search process, the population returns fitter solutions at each iteration, and ultimately the algorithm converges.
Convergence means that the global optimal was found, and the result is dominated by a unique solution (Harman, 2007).

GAs utilize crossover and mutation as principal operators to transform existing solutions into new solutions. The crossover operator is the most important (Konak et al., 2006); it combines two chromosomes, the parents, to create a new chromosome, known as an offspring. The parents are selected among the chromosomes with preference towards objective values. The offspring receive a good combination of genes that make the parent chromosomes set closer to the optimal solution (Konak et al., 2006).

The mutation operator operates at the gene level by randomly introducing changes into genes of a chromosome. In GA, mutation is based on a random mutation rate, which is the probability of changing a gene. The mutation probability rate depends on the length of the chromosome. The rate is usually small, indicating that the new chromosome is similar to the original chromosome after applying the mutation operation. Mutation has an important role in GA because it reintroduces genetic diversity into the population (Konak et al., 2006).

Reproduction refers to the selection of chromosomes that pass to the next generation. Generally, the probability of an individual survival for the next generation is regulated by its objective function values. The most popular selection procedures are proportional selection, ranking, and tournament selection (Goldberg & Holland, 1988).

3.2.3. Multi-objective GAs

GAs are a good population-based approach for solving multi-objective optimization problems. The capacity of GAs to search different areas of a solution space simultaneously allows
for finding a diverse set of solutions for difficult problems with non-convex (problems whose solutions will never converge to a global optimal), discontinuous, and multi-modal solutions spaces (Konak et al., 2006). GAs are the most popular metaheuristic approach to multi-objective design and optimization problems. GAs are evolutionary algorithms (EAs); they work with a population of solutions. Their goal is to move closer to the true pareto-optimal region. EAs are capable of finding multiple pareto-optimal solutions at each iteration (Deb et al., 2002).

The non-dominated sorting genetic algorithm (NSGA) proposed in (Zitzler & Thiele, 1999) was one of the first EAs. The NSGA has been described as having high computational complexity and lacking elitism (elitism involves copying a proportion of the fittest candidates, unchanged, into the next generation). NSGA-II is an improved version of NSGA (Deb et al., 2002).

3.2.4. NSGA-II

NSGA-II is a strong search approach based on Darwin’s evolutionist theory. It solves multi-objective optimization problems by applying non-dominated sorting (Deb et al., 2002). NSGA-II creates a large search space since it returns at each iteration a list of candidate solutions, known in the optimization field as local optimal solutions. The ultimate goal of the NSGA-II algorithm is to find a set of pareto-optimal solutions (Deb et al., 2002; Ghannem et al., 2017). The NSGA-II algorithm randomly creates the initial population $P_o$ of individuals. It uses genetic operators, such as crossover and mutation, to generate a child population $Q_o$ from the population of parents $P_o$. Both populations are combined, and the dominance principle, which defines which chromosomes are closer to the optimal solution, is used to select a subset of chromosomes to create the next generation. This process is repeated until the last iteration is reached as determined by a stopping criterion.
We explore the application of NSGA-II to the RTR problem because it performs better than other genetic algorithms for getting a more diverse set of solutions, and it converges near the true pareto-optimal set in comparison to other contemporary multi-objective evolutionary algorithms such as pareto-archived evolution strategy (Deb et al., 2002).

3.3. Related Work

Automatic RTR between software artifacts remains a challenge. In this research, we investigated the RTR problem as an optimization problem controlled by IR techniques. In this section, we provide an overview of IR techniques applied to the RTR problem and address existing research which has investigated the RTR problem using optimization algorithms.

3.3.1. IR techniques applied to the RTR problem

A number of IR inspired tools have been proposed to address the RTR problem (Hayes et al., 2007; De Lucia et al., 2005; Ali et al., 2011). RETRO, proposed in (Hayes et al. 2007), is a semi-automatic tool that implements the IR methods VSM and LSI to generate a requirement traceability matrix (RTM). ADAMS Re-Trace, proposed in (De Lucia et al., 2005), is a tool that implements LSI and supports identifying traceability links across different artifacts of the system. Trustace presented in (Ali et al., 2011) combines IR techniques and a data mining approach to create traceability links across the requirements and the source code of the system. Moreover, in many cases IR techniques have been used as fundamental technologies of semi-automatic approaches for extracting traceability links (Antoniol et al., 2002; Rodriguez & Carver, 2019; Antoniol et al., 2000; Antoniol et al., 1999; Hayes et al., 2003).
Application of IR probabilistic models to the RTR problem is presented in (Antoniol et al., 2000) which describes an approach to create and maintain traceability links between source code and requirements document. The approach works on code developed using object-oriented languages such as Java or C++, and it creates traceability links from classes of the source code to functional requirements. Results are validated over the Albergate data set which is a Java programmed system that was developed for students at the University of Verona (Italy) to help administer and manage small to medium sized hotels. Results show 50% average P and R (Antoniol et al., 2000).

An approach also based on IR probabilistic models applied to source code using C++ to trace classes to sections in the documentation manuals is given in (Antoniol et al., 1999). This research generates a language model from each section, and it transforms the textual content of the source code to create a system representation that minimizes the errors that could be introduced by coding styles such as mnemonic concatenation and abbreviations. The language model is compared with the generated system representation to match the sections where the concepts that the source code implements are described.

Application of VSM to the RTR problem can be found in (Hayes et al., 2003). This research uses a classical VSM algorithm to improve candidate link generation. Results indicate that even though the proposed VSM implementation does not return better values than analysts or other existing tools in terms of P and R, it is faster and does not require any keyword assignment.

An IR approach which uses both IR probabilistic and VSM techniques in two case studies to find traceability links from C++ source code to sections in the documentation manuals and Java code to functional requirement is found in (Antoniol et al., 2002). The research showed low values
of P and high values of R for both IR probabilistic and VSM when evaluated over the first case study (LEDA). The approach, when evaluated over a second case study (Albergate), returned slightly better results than for the previous case study, but still low values of P and high values of R of about 25% and 82%, respectively. It returned 10% P and 90% R on average for both of the evaluated IR techniques.

Mahmoud & Niu (2014) evaluated the performance of multiple semantic enabled IR methods such as VSM with thesaurus support (VSM-T), Part-of-Speech-enabled VSM (VSM-POS), LSI, latent Dirichlet allocation (LDA), explicit semantic analysis (ESA), and normalized Google distance (NGD) using three different datasets. The research aims to determine the methods which perform better to capture and present requirement traceability links in software systems. Methods are compared with the baseline approach VSM. Results indicate that explicit semantic methods (VSM-T, VSM-POS, ESA, and NGD) are a better alternative for recovering traceability links than latent methods (LSI and LDA). As part of the findings of this research, authors also establish that “considering more semantic relations in retrieval does not necessarily lead to a better tracing performance”, in some cases it introduces more noise to the data.

Another approach for studying multiple IR techniques when applied to the RTR problem, which is described in (Rodriguez & Carver, 2019), compares the effectiveness of different IR approaches as they are applied to the RTR problem. The study evaluated three IR approaches: IR probabilistic, VSM, and LSI. Results showed high P above 90% and low R for both VSM and IR Probabilistic. High P values inspired the use of IR techniques in our current research. The researchers postulated that low R results are because IR techniques were applied without assigning a threshold or using machine learning for automatizing the process. In that sense, the implementation of these IR methods returns for each requirement a single link with the highest
similarity value between the specific requirement and the class from the system source code. The approach does not handle cases where a requirement is related to more than one class. It outputs a list of links with at most one link per requirement, decreasing the R values.

The majority of the tools and approaches described above are time consuming or semi-automatic. Previous research in the area suggests that focusing purely on basic implementation of term matching techniques is unlikely to produce a precise and complete set of trace links if it is not combined with more intelligent approaches because it is an oversimplification of the cognitive steps required for the trace creation process (Cleland-Huang & Guo, 2014).

An approach using more intelligent trace retrieval algorithms presented in (Cleland-Huang & Guo, 2014) describes the study and generation of automated trace creation techniques based on a variety of algorithms, ranging from basic term matching approaches to more sophisticated expert systems. The approach, which is oriented to the creation of expert systems, indicates that development of more intelligent approaches for the RTR problem is needed.

In this research, we present an automated approach that combines IR techniques with optimization algorithms. We utilize the textual information in the requirements document and source code to identify possible matches that suggest the existence of links. The IR techniques are combined with the application of NSGA-II to achieve automatization.

3.3.2. Traceability link recovery as an optimization problem

We found only a few research efforts which studied the RTR problem as an optimization problem. Use of genetic algorithms for the automatic generation of traceability links is described in (Ghannem et al., 2016; Ghannem et al., 2017).
A mono-objective genetic algorithm approach which takes as input a set of requirements and the source code of the system and outputs a set of traceability links between the classes of the system and the requirements is described in (Ghannem et al., 2016). The solution is based on matching a requirement, using the requirement’s textual description, with one or many classes of the source code by comparing with textual elements of the source code such as identifiers, API documentation, and comments. The approach generated the traceability links between the requirements and classes in the source code with an average P of 86% and an average R of 83%. In contrast to (Ghannem et al., 2016), we use a multi-objective approach, but we utilize only information related to textual content of requirements document and the identifiers of the code elements to find the matches.

Another approach, presented in (Ghannem et al., 2017), uses NSGA-II and three objective functions, semantic similarity, recency of change measure, and frequency of changes measure. The results that were validated over three data sets show that P and R are above 80% on average. However, “if no information or only incomplete information is available about the times of change of the artifacts, the approach will not be able to take advantage of the recency of change and frequency of change objectives and optimize them” (Ghannem et al., 2017). Our study differs from (Ghannem et al., 2017) since we address the issue that available data sets for RTR studies often are not complete, and links need to be recovered using minimal information. We based our investigation on using system source code and the requirements document. We utilized two objective functions, Jaccard similarity and a weighted cosine similarity based on Term Frequency-inverse document frequency (TF-idf). Both are well known techniques for finding textual similarities.
We developed an approach for automatizing RTR which aims to decrease the complications of manual generation of traceability links. We explore the application of an IR-based NSGA-II approach to the RTR problem and study its potential benefits. We apply NSGA-II because it performs better for getting a more diverse set of solutions, and it converges near the true pareto-optimal set in comparison to another contemporary multi-objective evolutionary algorithm such as pareto-archived evolution strategy (Deb et al., 2002).

3.4. Adaptation of the IR-based NSGA-II to solve the RTR problem

We adapted NSGA-II to find the best trade-off for traceability link recovery using Jaccard similarity and weighted cosine similarity with TF-idf objective functions. The original implementation of NSGA-II is designed to work with decimal numbers that are randomly generated and evolved to the optimal solution through selection, mutation and crossover operators. However, our input is textual data formed by a set of requirements and source code (classes). Thus, we modified the generation of the population step, the data structures to adapt to our input data and some internal operators of the algorithm such as the mutation operator. We added modifications to the original NSGA-II, but we still applied all the operations and followed the pseudo-code defined for the original algorithm. The genetic operators of selection and crossover were applied without any logical modification, except for data structures modifications.

The first step in the NSGA-II algorithm is to generate an initial population. In the original implementation of the algorithm, the population is formed by decimal values vectors where each vector in the population is a chromosome, each decimal value in the vector is a gene, and multiple vectors form a population set. In contrast, in our implementation, the genes are pairs which contain exactly one requirement from the requirements document and a class from the system source code,
denoted by REQ and SCC, respectively. To manage the generation of the population, we modify original data structures at the implementation level to work with pairs of integers instead of decimal values. Integer pairs result by assigning a unique ID (integer number) to each element within the input files (requirements and classes), and we use those unique IDs to identify specific requirements or classes. The number of IDs is related to the number of requirements and classes in the requirements documents and the source code.

We randomly generate a population $P_0$ or initial population, regulated by the maximum size of an individual, called Max-size-I, which is a parameter that we introduced for our implementation. It controls the maximum number of pairs composing an individual $I$. During the execution, we randomly build the pairs for each individual, but our algorithm always verifies that the generated pairs are within boundaries which are defined to guarantee the textual analysis since we have a more limited search space than the original algorithm. Our lower bound is zero which is the minimum unique ID assigned, and the upper bound for REQs and SCCs depends on the maximum number of requirements and source code classes, respectively. Max-size-I depends on the number of requirements and source code classes in the data set. As a result, individuals can have different sizes depending on the data set. Out of boundary or nonexistent IDs will break the execution of the algorithm.

Figure 3.1 shows our mapping of the RTR terminology to the NSGA-II terminology; pairs represent genes, chromosomes represent individuals, and populations are made up of all the individuals.
As in the original NSGA-II version, we generated a population of RTR solutions regulated by the maximum size of the population, called $\text{Max-size-P}$. $\text{Max-size-P}$ denotes the number of individuals in the population. The $\text{Max-size-P}$ controls the number of individuals evaluated at each iteration of the algorithm. We create a population of candidate solutions which is transformed to achieve the best set of RTR solutions, known as the first front. The goal is to find the population that maximizes, as much as possible, the textual similarity based on Jaccard similarity across pairs (REQ, SCC) of each individual while favoring the weighed cosine similarity based on term frequency ($TF-idf$) across pairs of REqs and SCCs of each individual.

During each iteration $t$, we use genetic selection, crossover and mutation operators to generate an offspring population $Q_t$ from a parent population $P_t$. Next, we merge $Q_t$ and $P_t$ to create a global population $G_t$. We then use the objective functions to evaluate each solution $I$ in the population $G_t$. The generic operators of selection and crossover differ from the original NSGA-II only at the data structures level in order to adapt to the restrictions of our input and the differences in the population structure, but at the high level (logical implementation) they follow the same structure as the original NSGA-II. However, the mutation operator differs from the...
original algorithm at both, the structure and implementation levels. We present more detailed information in the remainder of this section.

Once values for the objective functions were calculated, we sort the solutions in order to return a list of non-dominated fronts $F(F_1, F_2, ...)$, where $F_1$ is the set of non-dominated solutions, and $F_2$ is the set of solutions dominated only by solutions in $F_1$, and so on. We then verify the set of nondominated fronts starting from front $F_1$ to $F_i$ to build the next population $P_{(t+1)}$. Generally, the number of solutions in all sets from front $F_1$ to $F_i$ is greater than $Max-size-P$. We use the crowded-comparison operator defined in (Deb et al., 2002) to sort the solutions of the front $F_i$, then we select the first $Max-size-P$ solutions to ensure that we choose exactly $Max-size-P$ solutions.

The crowded-comparison operator uses non-dominated ranking which involves sorting a population $P$ based on the non-domination of individuals and the crowding distance which provides an estimate of the density of solutions surrounding that solution (Deb et al., 2002). Crowded-comparison and crowding distance are two techniques to ensure pareto-dominance and solution diversity (Deb et al., 2002). We defined termination criterion as the maximum number of generations ($Max-G$). The output is the set of best solutions, which is the set of solutions in the pareto-front of the last iteration.

3.4.1. Individual representation

We represent a candidate solution using a list of (REQ, SCC) pairs where REQ denotes the ID of the requirements and SCC denotes the ID of the source code class, represented by numerical values (integers). Table 3.1 shows an example of this representation. In contrast to the original implementation of NSGA-II, we worked with integer pairs instead of single vectors of decimal numbers.
Table 3.1. Representation of an NSGA-II individual.

<table>
<thead>
<tr>
<th>REQ</th>
<th>SCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>R115</td>
<td>SCC46</td>
</tr>
<tr>
<td>R117</td>
<td>SCC48</td>
</tr>
<tr>
<td>R121</td>
<td>SCC93</td>
</tr>
</tbody>
</table>

3.4.2. Selection operator for genetic algorithms

The most popular selection methods to choose the best individuals for the next generation are roulette wheel selection, boltzman selection, tournament selection, and rank selection (Deb et al., 2002). The NSGA-II uses “binary tournament selection to generate a child population $Q_t$” (Deb et al., 2002). The child population $Q_t$ “is the set of individuals that will go through the crossover and mutation operators, and they are selected from parents’ population $P_t$” (Deb et al., 2002). The binary tournament selection consists of running several tournaments. We formed each tournament by randomly selecting two individuals from the population $P_t$. This selection enables that all individuals of the population have equal chances of selection, thus preserving diversity. The selection operator is implemented as in the original NSGA-II; however, we modified the data structures at the implementation level to adjust to the representation of our input data.

3.4.3. Mutation and crossover operators for genetic algorithms

The mutation operator randomly changes one or more pairs of an individual. We applied mutation based on a randomly determined mutation factor. Once we select an individual, the mutation operator randomly selects one or more pairs (REQ, SCC) of the individual and replaces the pairs with other different pairs randomly chosen from the solution space.
The original version of the algorithm mutates the genes by applying mathematical operations which transform the original values (genes). However, values of genes are less restricted in the original version; mathematical operations often return decimal values which represents a disadvantage in our specific input. In that sense, we adapted the mutation operator. Once we selected the pairs to be mutated, we performed the mutation by generating new random pairs which need to be different from the original pairs to add diversity to the execution. We controlled the random generation of pairs to ensure that it returns integer values inside the IDs boundaries by limiting the possible random numbers to falling between the defined lower and upper bounds, which are related to the number of requirements and source code classes in the system. Figure 3.2 shows the pairs \((R123, SCC68)\), and \((R121, SCC68)\) mutated and replaced by the pairs \((R124, SCC92)\), and \((R137, SCC78)\), respectively.

![Mutation](image)

**Figure 3.2.** Results of application of mutation operator.

For the crossover operator, we maintained the strategy used in the original NSGA-II, and the only variation is at the implementation level to handle the new data structures. We applied tournament selection to select individuals (Deb et al., 2002). Then, we applied a double, random, cut-point crossover to create two offspring \(I_1^+\) and \(I_2^+\) from the selected parents \(I_1\) and \(I_2\). This crossover operation selects two individuals parents \(I_1\) and \(I_2\), and swaps genes between the two individuals, producing two children \(I_1^-\) and \(I_2^-\). The crossover happens according to a probability
based on a crossover factor chosen randomly. Figure 3.3 shows the implementation of the crossover operation.

3.4.4. Objective functions for the evaluation of individuals

We use textual information in requirements and source code classes to find possible matches or traceability links. We assumed that the vocabulary in the requirements is similar to the one used for the source code classes textual content, and we established semantic similarity based on that assumption. We used the textual content in requirements to generate queries that are compared to elements in the source code classes such as identifier names (classes, methods and variables) which hold the semantic information in the source code class. A high similarity across text associated to a specific REQ and a SCC suggests a strong relation among the two components which we interpret as a possible traceability link.

When modeling the objective functions that have to be maximized for NSGA-II, we applied two popular measurements for finding textual similarities, SemSimJS and SemSimTF. SemSimJS is based on Jaccard similarity, and SemSimTF considers term frequency by first applying $TF-idf$ to create a weighted representation of the text from REQs and SCCs and then
computes cosine similarity. Application of these objective functions leads the algorithm to the optimal solution represented by the set of links which, as a set, returns the higher similarity value.

We computed SemSimJS between a requirement and an artifact source code class by applying the Jaccard similarity operation. The documents were parsed and split to extract single words or terms; then, the similarity is represented by the ratio of the intersection and the union of words in both documents. The objective function value associated with the Jaccard similarity for an individual \( I \) is the average of the Jaccard similarity value for each pair \((\text{REQ}_i, \text{SCC}_i)\) in the individual \( I \). Eq. (3.2) defines SemSimJS, where \( m \) is the number of pairs in the individual.

\[
\text{SemSimJS}(I) = \frac{m}{\sum_{i=1}^{m} \left( \frac{\text{REQ}_i \cap \text{SCC}_i}{\text{REQ}_i \cup \text{SCC}_i} \right)}
\]  

(3.2)

We computed SemSimTF by applying \( TF-idf \) to transform the textual content associated with \( \text{REQs} \) and \( \text{SCCs} \) into weighted vectors that support the application of cosine similarity to generate a similarity factor. The vectors are of length \( x \), where \( x \) is the number of different terms in the documents. In \( TF-idf \), \( TF \) stands for term frequency and \( idf \) is the inverse document frequency factor. SemSimTF considers whether a term \( T \) extracted from a requirement is present in a source code class, as well as the number of occurrences of term \( T \) in that source code class and in all other (requirements and source code classes). For \( TF-idf \), it is relevant that a term that occurs in a few documents is likely to be more important than a term that appears in most or all documents. We computed the value of the objective function SemSimTF that gives the term frequency for an individual \( I \) as the average \( TF-idf \) weighed cosine similarity values for each pair
(REQi, SCCi) in the individual I. Equation (3.3) defines the SemSimTF, where $m$ is the number of pairs in the individual.

\[
TF_{(t,d)} = \frac{\text{Number of terms appearing in the document (d)}}{\text{Total number of terms in the document (d)}}
\]

\[
idf_{(t,D)} = \log \left( \frac{\text{total number of documents (D)}}{\text{Number of documents with term (t) in it}} \right)
\]

\[
TFidf_{(t,d,D)} = TF_{(t,d)} \cdot idf_{(t,D)}
\]

\[
\text{SemSim}(I) = \sum_{i=1}^{m} \text{CosineSimilarity} \left( \text{Weight}_{TF-idf}(\text{REQ}_i, \text{SCC}_i) \right)
\]

Equations (3.2) and (3.3) are our objective functions. We find the optimal solution by optimizing the values of both equations.

3.5. **NGSA-II implementation and experimental setup**

3.5.1. **NSGA-II implementation for the RTR problem**

We implemented our adaptation of the IR-based NSGA-II algorithm using Python 3.6. We use three open-source projects, Event Based Traceability (EBT), Albergate and Tour Guide System (eTour), to test our approach. All data sets were downloaded from http://coest.org/. The CoEST website, found at http://coest.org/index.php/what-is-traceability, provides data sets for traceability link recovery research.

We selected EBT, Albergate and eTour because they represent small and medium sized data sets with the needed structure for our research. The data sets include the requirements documents, source code divided by classes, and traceability links from the requirements document to the classes of the system. We require a data set that has well-structured requirements document presented as a list, source code divided in individual classes, and a list of traceability links which
we use to validate our findings. Even though several data sets are found in CoEST, EBT, Albergate and eTour are the only three we found with the required structure.

The EBT data set, developed in Java, consists of 40 requirements, 50 Java source code classes, and 98 trace links from requirements to classes (Cleland-Huang et al., 2003). Albergate is a software system developed in Java and designed to implement operations such as room reservation and bill calculation required to administer a small/medium size hotel. Final year students at the University of Verona (Italy) developed the system that has 17 requirements, 55 source code classes, and 54 trace links from requirements to classes (Antoniol et al., 2002). The eTour data set, which is also developed in Java, includes 58 use cases used to extract the requirements, 116 source code classes, and 308 trace links from requirements to classes (Poshyvanyk et al., 2011).

We divided our implementation into two main phases. The first phase transforms the raw data into meaningful information for use with the NSGA-II algorithm. We manually pre-process the raw data to extract meaningful information to be evaluated for the NSGA-II algorithm. Figure 3.4 illustrates the pre-processing which removes stop words, removes punctuation, transform capital to lower case letters, and performs stemming to get word roots. The source code preprocessing keeps only the identifiers names, including variables, methods, and class.

![Figure 3.4. Preprocessing implementation.](image)
The second phase implements the genetic operators and evaluates the objective functions, iterating until the stopping criterion \( \text{Max-G} \) is reached and the generation that maximized the objective functions is found. Figure 3.5 illustrates the process.

![Figure 3.5. NSGA-II implementation.](image)

### 3.5.2. NSGA-II set up

The parameter \( \text{Max-size-I} \), which is associated with the number of requirements and classes in the data set, was set to three times the number of classes in the data set. To establish a reasonable upper bound for the algorithm, we assumed that a class is related to at most three requirements. In that sense, the maximum number of pairs returned \( \text{Max-size-I} \) had to be larger than the number of requirements and at least three times larger than the number of classes defined in the data set. We made this assumption to increase the likelihood of finding all the possible links. \( \text{Max-size-I} \) was set to 150 for the EBT and Albergate data sets and 348 for the eTour data set based on experimentation on the size of the parameter.

We defined parameters \( \text{Max-size-P} \) and \( \text{Max-G} \) based on experimentation to achieve convergence. Once the algorithm converges, since there is no possibility of further improvements,
it repeats the same output at each future generation. To avoid having the same output multiple times, we defined $Max-G$ to be the smallest number that allows convergence. To achieve convergence faster, we assigned a large value to $Max-size-P$ to evaluate more individuals at each iteration.

Parameters $Max-size-P$ and $Max-G$ were set to 150 and 100 respectively for EBT, 150 and 50 respectively for Albergate, and 300 and 120 respectively for eTour. We assigned the crossover probability factor to 0.5 and the mutation probability parameter to 0.3. We selected a high mutation rate to avoid premature convergence since a higher rate enables diversification of a population.

The output is a generation that maximizes the objective functions represented by two lists of numbers $([n_0, n_1, \ldots, n_n], [n_0, n_1, \ldots, n_n])$, where the first list corresponds to the requirements $ID$ and the second list corresponds to the source code class $ID$. Figure 3.6 shows example output.

![Figure 3.6. NSGA-II output for EBT data set.](image)
3.6. Experimental results

3.6.1. RQ3-1: Can the multi-objective genetic algorithm NSGA-II controlled by IR-based objective functions assist automatic RTR?

To address answer RQ3-1 which aims to determine whether a genetic algorithm (NSGA-II) controlled by IR-based objective functions is feasible to assist automatic RTR, we evaluated our results based on P and R. P is a measure of exactness or quality calculated as the total number of correct links returned divided by the total number of links found. R is a measure of completeness or quantity calculated by the division of the number of correct links found by the number of correct links in the data set. High P and R rates are desirable. P and R are defined by Eq. 3.4 and Eq.3.5, respectively.

\[
P = \frac{TP}{TP+FP} \in [0,1] \\
R = \frac{TP}{TP+FN} \in [0,1]
\]

To find the values for P and R associated to the optimal solution, we identified true-positive (TP) values, false-positive (FP) values, and false negative (FN) values. TPs are pairs correctly identified, FPs are pairs that are not correct and should not be in the optimal solution (output) but are found in the solution. FNs are those pairs that should have been identified as correct but are not found and are not in the output.

The NSGA-II algorithm is based on an initial random population. As a result, the output has fluctuations with different executions. We iteratively executed our implementation 10 times for each data set. We computed the mean to find P and R based on the i iterations over each data set to help account for fluctuations in the outputs. The mean values of P and R are defined by Eq. 3.6 and Eq. 3.7, respectively.
Mean\(P = \frac{\sum_{i=0}^{10} P_i}{10}\) (3.6)

Mean\(R = \frac{\sum_{i=0}^{10} R_i}{10}\) (3.7)

Table 3.2 shows the number of correct links retrieved, total number of distinct links retrieved, and P and R for each of the 10 iterations for the EBT data set. It also shows the mean P and R values. Both P and R mean values range above 77%, representing an acceptable result on a medium size data set.

**Table 3.2. P and R for 10 iterations over EBT data set.**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
<td>101</td>
<td>74%</td>
<td>76%</td>
</tr>
<tr>
<td>2</td>
<td>84</td>
<td>99</td>
<td>85%</td>
<td>86%</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>107</td>
<td>75%</td>
<td>82%</td>
</tr>
<tr>
<td>4</td>
<td>67</td>
<td>80</td>
<td>84%</td>
<td>68%</td>
</tr>
<tr>
<td>5</td>
<td>71</td>
<td>89</td>
<td>80%</td>
<td>72%</td>
</tr>
<tr>
<td>6</td>
<td>68</td>
<td>96</td>
<td>71%</td>
<td>69%</td>
</tr>
<tr>
<td>7</td>
<td>73</td>
<td>90</td>
<td>81%</td>
<td>74%</td>
</tr>
<tr>
<td>8</td>
<td>74</td>
<td>105</td>
<td>70%</td>
<td>76%</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>102</td>
<td>78%</td>
<td>83%</td>
</tr>
<tr>
<td>10</td>
<td>82</td>
<td>104</td>
<td>80%</td>
<td>84%</td>
</tr>
<tr>
<td>Mean</td>
<td>75.5</td>
<td>97.3</td>
<td>77.8%</td>
<td>77%</td>
</tr>
</tbody>
</table>

Figure 3.7 shows a graphical representation of the results obtained for P and R at each iteration of the EBT data set. We observed higher values of P in comparison to R at each iteration, and results express differences across P and R ranging within 2% and 17%. Results suggest that our approach when evaluated over EBT returned a precise list of links associated with the P values, and it generated acceptable R values.
Figure 3.7. Values of P and R for 10 iterations over EBT data set.

The best P value in EBT occurred in iteration 2 where the approach returned 84 correct links out of the 99 retrieved links, representing 85% P; the best R value occurred in iteration 2 which finds 84 correct links out of the 98 links in the original data set, representing 86% R.

Table 3.3 shows the values of P and R for 10 iterations over the Albergate data set. Similar to EBT, the table shows the number of correct links retrieved, total number of links retrieved, the values of P an R, and their mean values. For Albergate the mean values for P and R are 67% and 83%, respectively.

Figure 3.8 shows P and R at each iteration over the Albergate data set. In contrast to EBT, for Albergate we observed higher values of R, in comparison to P, at each iteration. Differences across P and R values range between 10% and 25% which represents a larger gap compared to the EBT results. Results suggests that our approach when evaluated over the Albergate returned a more complete list of links associated with the R values, but generated lower P values. Even though the approach found most of the links in the original data set, it generated many FPs. A possible explanation is because of the value of $Max-size-I$ assigned in the setup of the experiment. For Albergate, $Max-size-I$ is 150, but this data set only has 54 links. A large gap between the number
of links and the real number of links implies an increment in the number of FPs, which affects the P values.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42</td>
<td>63</td>
<td>67%</td>
<td>76%</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>74</td>
<td>60%</td>
<td>83%</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
<td>66</td>
<td>71%</td>
<td>87%</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>62</td>
<td>71%</td>
<td>81%</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>63</td>
<td>71%</td>
<td>71%</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>66</td>
<td>73%</td>
<td>89%</td>
</tr>
<tr>
<td>7</td>
<td>45</td>
<td>73</td>
<td>62%</td>
<td>83%</td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>70</td>
<td>60%</td>
<td>78%</td>
</tr>
<tr>
<td>9</td>
<td>49</td>
<td>71</td>
<td>69%</td>
<td>91%</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
<td>68</td>
<td>69%</td>
<td>87%</td>
</tr>
<tr>
<td>Mean</td>
<td>45.4</td>
<td>67.6</td>
<td>67.3%</td>
<td>82.8%</td>
</tr>
</tbody>
</table>

The best precision value in Albergate occurred in iteration 6 with 48 correct links out of the 66 retrieved links, representing 73% P; the best R value occurred in iteration 9 which recovered 49 corrects links out of the 54 links in the original data set, representing 93% R.

Table 3.4 shows the values of P and R for 10 iterations over the eTour data set. Similar to EBT and Albergate, the table shows the number of correct links retrieved, total number of links retrieved, the values of P an R, and their mean values. For the eTour which is a larger data set, the mean P is 83% and the mean R is 63%.
**Figure 3.8.** Values of P and R for 10 iterations over Albergate data set.

**Table 3.4.** P and R for 10 iterations over eTour data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>187</td>
<td>233</td>
<td>80%</td>
<td>61%</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>231</td>
<td>87%</td>
<td>65%</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>230</td>
<td>84%</td>
<td>63%</td>
</tr>
<tr>
<td>4</td>
<td>205</td>
<td>225</td>
<td>91%</td>
<td>67%</td>
</tr>
<tr>
<td>5</td>
<td>202</td>
<td>227</td>
<td>89%</td>
<td>66%</td>
</tr>
<tr>
<td>6</td>
<td>185</td>
<td>230</td>
<td>80%</td>
<td>60%</td>
</tr>
<tr>
<td>7</td>
<td>182</td>
<td>248</td>
<td>73%</td>
<td>59%</td>
</tr>
<tr>
<td>8</td>
<td>195</td>
<td>240</td>
<td>81%</td>
<td>63%</td>
</tr>
<tr>
<td>9</td>
<td>203</td>
<td>238</td>
<td>85%</td>
<td>66%</td>
</tr>
<tr>
<td>10</td>
<td>197</td>
<td>237</td>
<td>83%</td>
<td>64%</td>
</tr>
<tr>
<td>Mean</td>
<td>195</td>
<td>233.9</td>
<td>83%</td>
<td>63.4%</td>
</tr>
</tbody>
</table>

Figure 3.9 illustrates P and R at each iteration over the eTour data set. As opposed to the Albergate and following the EBT pattern, we observed higher values of P in comparison to R at each iteration. Differences across P and R values range between 20% and 24%. Results suggest that our approach when evaluated over eTour was more precise since it returned several correct
links out the total number of links recovered without generating a high number of FPs. However, the results did not contain a complete list of links because the number of FNs was high, which affects the R values.

![Figure 3.9](image.png)

**Figure 3.9.** Values of P and R for 10 iterations over eTour data set.

For eTour, the best P value was returned in iteration 4 which recovered 205 correct links out of the 225 links retrieved, generating 20 FPs. The best R value was returned also in iteration 4 which returned 205 correct links out of the 308 links in the original data set. In addition, 103 FNs were found.

Figure 3.10 shows the performance for EBT, Albergate and eTour data sets based on mean values. The approach returned opposite results for Albergate, in contrast to EBT and eTour, returning low P and high R when evaluated over Albergate, and high P and lower R when evaluated over EBT and eTour. According to the mean values, our approach performed better in terms of P when evaluated over eTour, reaching 83% P, and it had better performance in terms of R over Albergate, returning 86% R. The lowest value of R was observed in eTour, and the lowest value of P was observed in Albergate. For EBT, returned values are acceptably high for both P and R.
Figure 3.10. Mean values of P and R for 10 iterations over EBT, Albergate and eTour data sets.

For EBT the mean P value was approximately 78%, and the mean R value was approximately 77%, representing higher preciseness than completeness. However, the difference between the two metrics was small. We recovered a mean of 76 correct links out of the 98 links in the original EBT data set, and the mean of total links retrieved over EBT was 97, which is very close to the value of 98 links in the original data set. The mean total links of 97 explains the results for P and R and the small difference among the two values since the numbers of FPs and FNs was low.

In contrast, in the Albergate data set in which the number of links retrieved was higher than the number of links in the original data set, we observed higher completeness that preciseness. When tested over Albergate, our approach returned mean values for P and R of 67% and 83%, respectively. It retrieved a mean of 45 correct links out of the 54 links in the original data set, but it returned a mean total of link of 68 which explains the low P value since it was affected by the increment in the number of FPs.

Opposite to Albergate and similar to EBT, for eTour the P value was approximately 83% and R value was approximately 63%, which means that results were more precise and less
complete. The eTour results follow the pattern of EBT in term of P and R values; however, the difference between the two metrics was larger. For eTour, we found a mean of 195 correct links out of the 308 links in the original data set, which translates to an acceptable value for P. Additionally, results suggest that the mean of distinct links when evaluated over eTour was 234, which is smaller than the 308 links in the original data set representing an increment in the number of FNs returned, which affected the R values.

The results are highly related to the values of the parameters involved in the algorithm execution. We defined the parameter values based on experimentation. However, we hypothesize that values of P and R might be further improved by changing experiment settings. Values such as individual size \((\text{Max-size-I})\), population size \((\text{Max-size-P})\), maximum number of generations \((\text{Max-G})\), mutation rate and crossover rate could be modified with more experimentation.

These results are encouraging for applying an intelligent approach to the RTR problem. The result for RQ3-1 is positive; the application of a metaheuristic algorithm to assist the RTR produced promising results for P and R. It helps to enable automatic RTR by building sets formed by a combination of different pairs \((\text{REQ, SCC})\) at each generation.

Every generation is formed based on the pairs with higher similarity values; these pairs with higher similarity go from one generation to another. The automatization is based on forming the best generations at each iteration and stopping when a criterion value is met. This research complements IR approaches by adding the genetic components that create the best candidate solutions at each iteration.
3.6.2. RQ3-2: How effective is the application of a MS approach to the RTR problem in contrast to IR techniques?

To address RQ3-2 regarding the effectiveness of the results, we compared our results in terms of P and R to those obtained in (Rodriguez & Carver, 2019) based on the basic implementation of IR techniques of IR probabilistic, VSM and LSI adapted to the RTR problem. In (Rodriguez & Carver, 2019), the EBT and eTour data sets were used to evaluate findings, which facilitates the comparison to our approach since we also used the same data sets to evaluate our results. Figure 3.11 show those results for P and R for EBT and eTour data sets.

The experimental results suggest that IR probabilistic and VSM returned high values of P but low R values in contrast to our approach based on NSGA-II which outperformed the implementation of the IR techniques for R values while maintaining a good but not as high P value. We observed a difference between P and R when applying the IR techniques and a reduced gap in P and R when applying our adaptation of the NSGA-II algorithm.

![Figure 3.11. Comparison of the P and R values of the NSGA-II algorithm and IR techniques over EBT and eTour.](image)

65
We assumed that values of P and R are equally important. To define the effectiveness among approaches, we use the F1 measure which is the harmonic mean, or weighted average, of P and R scores. We use F1 as a measurement to seek a balance between P and R, and it is calculated according to Eq. (3.8).

\[
F1 = 2 \frac{Precision \times Recall}{Precision + Recall}
\]  

(3.8)

Figure 3.12 illustrates the F1 values for NSGA-II and IR techniques of IR probabilistic, VSM, and LSI calculated over EBT and eTour respectively. Our experimental results indicated that although our implementation of the NSGA-II algorithm achieved slightly lower P values, our intelligent approach found an improved balance between P and R in contrast to the IR techniques.

![Figure 3.12](image)

**Figure 3.12.** F1 values of the NSGA-II algorithm and IR techniques over EBT and eTour.

The NSGA-II algorithm applied to the RTR problem allows working with a population of links. It automatically generated different lists of pairs (REQ, SCC) which are evaluated and modified based on the NSGA-II parameters to increase and maximize the objective functions. Our NSGA-II implementation returns the list of (REQ, SCC) pairs (links) that, as a set, maximizes the objective functions. The resultant list of (REQ, SCC) pairs is a complete list with the highest
number of positive matches. In contrast, the implementation of the IR techniques (Rodriguez & Carver, 2019) only found for each requirement REQ the class SCC with the highest probability of being related to the specific requirement. Thus, they found only a single pair for each requirement, which affected the R value.

Our research is inspired by the promising P results obtained by the application of IR techniques to the RTR Problem. We control the MS by applying IR techniques, but we include the genetic algorithm components as a mechanism to allow an automatic smart search that leads to the optimal solution represented by the list of links with the highest number of matches (correct links).

In comparison to the baseline approach presented in (Rodriguez & Carver, 2019), our implementation is more effective and a better alternative for addressing the RTR problem. Our approach allows automatization, and it returned high F1 values according to P and R, which increases the trust factor in the approach.

Moreover, for further analysis, we compared our results to results obtained in the research described in (Ghannem et al., 2017) which applies the NSGA-II algorithm to RTR. The findings over the eTour and Albergate, two of the data sets used for our analysis, facilitate the comparison. Figure 3.13 shows the results.

The approach in (Ghannem et al., 2017) obtained better results in most of the cases, except for the R values when evaluated over the Albergate data set; however, our approach can be applied to cases where only minimal information about the system is available. The approach presented in (Ghannem et al., 2017) is based on three objective functions, and it requires the requirements document, the source code, information about change history, and frequency of changes. Our approach only requires the requirements document and the source code.
Figure 3.13. Comparison of P and R of our approach and Ghannem et al., 2017.

Research questions RQ3-1 and RQ3-2 are intended to determine if the application of an optimization algorithm to the RTR problem is feasible and how effective our approach is in comparison to baseline IR techniques. Experimental results based on the metrics of P and R give positive answers to both of the questions. Results encourage the application of MS combined with IR techniques to automate the RTR problem.

3.7. Threats to validity

We have designed our approach with a goal to reduce threats to validity; however, there are some threats that we could not eliminate. An internal threat to validity is the assumption that the requirements document and the source code follow guidelines. We assume that the requirements document does not contain abbreviations and that the source code was written using good programming techniques, including that the names of identifiers (classes, methods and variables) are meaningful, and the source code is well documented. If the input data does not satisfy these assumptions, our approach will not be able to establish relationships between the documents as it is based on a semantic analysis.
We assume that the requirements document is written in a way that we can identify a clear list of individual requirements from the document which are associated with a unique ID, and that the source code is written using a well-structured object-oriented approach that allows identification of individual classes or code fragments which are associated with a unique ID. Our approach is designed for object-oriented code where individual classes can be identified. Absence of object-oriented code would limit the application of our approach, since it would be challenging to identify fragments of code related to specific requirements.

An external threat to validity relates to the quantity of data sets. The availability of more data sets that include requirements document, source code divided in classes and traceability links used for validation would strengthen the evaluation. Our use of only three data sets affects the generalizability of the study. More available data sets are needed for more experimentation to ensure that our approach could be broadly applied to the RTR problem.

Given that some of the parameters in our NSGA-II implementation were defined based on experimentation, the parameters could also affect the generalizability of the study. A deeper analysis based on a larger number of data sets is needed to define patterns to help in the definition of the parameter values. Even though the parameters controlling mutation, crossover, and individual size could be applied to all data sets, more standard values for parameters such as population size and number of generations are needed. However, most parameters from the related research are based on experimentation.

3.8. Conclusions

We addressed the RTR problem as a search-based optimization problem. The popularity and efficiency of metaheuristic algorithms were inspirations for our research. Search-based
optimization algorithms iteratively evaluate sets of possible solutions to find the set that maximizes the objective functions. In the RTR problem, requirements and source code are the main inputs. The output is a set of traceability links from requirements to source code that maximizes the IR-based objective functions defined by textual similarity using Jaccard similarity and weighed Cosine Similarity with Term Frequency (TF-idf). IR techniques are good techniques for finding relationships because both requirements and source code are expressed in text and both hold semantic information. The final set of links is generated by the random and iterative generation of sets of lists of requirements and source code (REQ, SCC) evaluated based on the objective functions. Values for P and R using our implementation of the NSGA-II algorithm adapted to the RTR problem are above 77% in EBT, 70% in Albertage and 70% in eTour on average according to the F1 values.

Additionally, our intelligent approach finds an improved balance between P and R, in comparison to another RTR investigation using only IR techniques (Rodriguez & Carver, 2019). For our approach, the F1 values used to find a balance across P and R are 77% and 72% for EBT and eTour respectively which are higher than the F1 for IR-probabilistic, VSM and LSI where values range from 1% to 50% over both studied data sets.

In comparison to other approaches using the NSGA-II algorithm, we observed that even though we obtained lower values of P and R, our approach can be used broadly in cases were only minimum information about the system is available. Yet, our implementation returns good results of approximately 77% and 76% for P and R respectively on average.
We plan to expand our study to include additional data sets, different objective functions and additional intelligent algorithms with the goal of improving the RTR problem and achieving automatization as well as accuracy.
4. AN IR-BASED ARTIFICIAL BEE COLONY APPROACH FOR TRACEABILITY LINK RECOVERY

Multiple software development activities, including change impact analysis, requirements validation, maintenance and evolution of software, benefit from adequate requirements traceability link recovery practices. Requirements traceability link recovery increases the overall quality of software products; however, companies often are unsuccessful implementing them due to lack of communication and strict deadlines. Several approaches for semi-automatic link recovery across requirements and source code have been developed, in which textual analysis and information retrieval (IR) techniques are the baseline, but there is a need for methods that further enable the automatic generation of links. For aiming automatization in the process, we investigate the requirements traceability recovery problem as a combinational problem that should be solved by an optimization approach. In this contribution, the requirements traceability recovery problem is studied as a big search space formed by multiple pairs (requirements and source code classes), where requirements are associated to code elements. A requirement can be related to more than one class, and a class linked to more than one requirement. The artificial bee colony (ABC) algorithm is adapted for searching for the solution that maximizes an objective function which is based on semantic similarity, and it is calculated by a weighted cosine similarity where weighs for each term in the textual content of requirements and source code are defined according to the term frequency ($TF-idf$), which is a popular method for finding text similarities. Our application of the ABC algorithm which uses an IR-based objective function for finding textual similarities returned high precision and recall values for the recovered links. Evaluation over three open source data sets demonstrates the effectiveness of the proposed approach.
4.1. Introduction

Requirements Traceability Recovery (RTR) is a software engineering task which establishes a relationship between software artifacts and requirements. During software development, a main objective is to guarantee that the designed system matches the set of actual requirements. From small to complex systems, requirements are continuously changing due to many factors. Thus, ensuring the correct implementation of the requirements resembles a difficult task (Faiz et al., 2006). The ability to follow the life of requirements throughout the development process, and to guarantee that they are correctly implemented increases significantly the quality of the software (Mills & Haiduc, 2017).

RTR has become an important topic in software engineering. Requirements and other artifacts including source code, design documents and test cases are associated through traceability links. In this research, we concentrate on generating links between requirements and source code because requirements to code traceability links are important to determine the parts of code that implement each requirement. These links provide support in a variety of activities such as change impact analysis, testing, requirements verification, program comprehension and reuse (Ghannem et al., 2017). In this context, traceability has high importance, particularly in safety-critical software system, where missing requirements could lead to disastrous consequences (Faiz et al., 2006). Although traceability is important to software development, the effort needed to manually establish and maintain traceability links is high.

Researchers have investigated a number of techniques for the automation of RTR. Techniques range from IR, ontological approaches (Zhang et al., 2008), machine learning methods, and other tools such as REquiremnet TRacing On-target RETRO (Hayes et al., 2007),
ReqSimile (och Dag et al., 2005), Poirot: TraceMaker (Lin et al., 2006), ReqAnalyst (Lormans & Van Deursen, 2006), TraceViz (Marcus et al., 2005), and ADAMS Re-Trace (De Lucia et al., 2005). However, most of the proposed tools are time-consuming or semi-automatic and require user manipulation.

IR techniques support RTR. Links across source code and requirements document are generated by applying IR methods to match a query of key words extracted from the source code, focusing on source code identifiers names (methods, variables, and classes) with a set of relevant words extracted from the requirements document (Faiz et al., 2006; Mahmoud & Niu, 2014). IR techniques are based on the textual component of requirements and source code. Similar terms suggest a strong connection between a specific requirement and fragments of the source code (class) (Ali et al., 2011). An IR approach typically assumes that the source code was written using good programming style. In our work, in order to find possible matches or similarities between requirements and classes of the source code, an assumption is that meaningful names were assigned to the code components.

A feasible solution for the RTR problem in terms of similarities across requirements and source code can be accomplished by using an IR approach. Even though IR techniques are semi-automatic solutions, they have become the baseline for other methods aiming for fully automatic RTR. Automating RTR implies the generation of traceability links without requiring an analyst to manually review all the requirements (Deb et al., 2002). The research literature suggests that IR techniques combined with optimization algorithms are good alternatives for automation RTR (Ghannem et al., 2016; Ghannem et al., 2017). While several solutions for RTR have been proposed, there is a need for techniques that produce more accurate results which depend less on human intervention (Mahmoud et al. 2012).
In this work, we developed a method to assist the automation of RTR by investigating it as a combinatorial optimization problem which evaluates the association between requirements and the source code. We applied a modified version of the swarm intelligence algorithm Artificial Bee Colony (ABC).

The ABC is an optimization algorithm proposed by Karaboga (2005). It is based on the intelligent behavior of a honeybee swarm. Swarm intelligence (SI) is a sub-field of computational intelligence (CI) focused on developing bioinspired multi-agent intelligence systems. It uses the collective behavior of natural agents such as bees to create the algorithms (Karaboga, 2005). These algorithms have proven to be efficient for solving real-world problems. Mathematically speaking, to solve a real-world optimization problem using CI algorithms, a mathematical representation of the problem is needed. This representation is the objective function, which resembles a mathematical rule that describes the problem and its decision variables (Karaboga, 2014; Karaboga, 2007).

Our approach takes a set of requirements and the source code of a software system as the input. It generates a set of links from the requirements to the source code as the output. The output is the set that maximizes the objective function. The objective function is based on the application of the Vector Space Model (VSM) and uses a weighted cosine similarity. The weights are based on the frequency $TF-idf$ to generate the vectors. The objective function is defined by the sum of all the similarities’ percentages resulting from evaluating each link in the set. A higher value for the objective function translates to more matches. The set of links that maximizes the objective function is defined as the optimal solution.
We address the following research questions:

RQ4-1: Can the SI algorithm ABC and an IR-based objective function assess automatic RTR?

RQ4-2: How effective is the application of CI algorithm to the RTR problem in contrast to IR techniques?

RQ4-3: How effective is the application of a SI algorithm to the RTR problem in contrast to genetic algorithms?

The remainder of this paper is structured as follows: section 4.2 contains background information, section 4.3 discusses related work, section 4.4 describes the methodologies for the ABC algorithm adapted to the RTR, section 4.5 explains the implementation of ABC, section 4.6 contains the experimental results, and section 4.7 presents conclusions and future work.

4.2. Background

We investigate RTR as an optimization problem. We implement the SI algorithm ABC to generate links across a requirements document and the source code. Algorithms such as the ABC fall into the CI category, and they are controlled by an objective function which in our research has to be maximized to find the optimal solution. We review the basic concepts of CI and SI algorithms.

4.2.1. Computational intelligence (CI) algorithm

CI is an emerging area of investigation commonly denoted as Artificial intelligence (AI). CI is defined as “the study of the design of intelligent agents where an intelligent agent is a system
that perceives its environment and then takes actions to maximize its chances of success” (Karaboga, 2014). Even though CI methods are considered AI, they differ from AI principally because CI utilizes sub-symbolic knowledge processing, and classical AI utilizes symbolic approaches (Karaboga, 2005). CI provides a number of nature-inspired computational methodologies to solve complicated problems with applications in the real-world.

CI includes neural networks (NNs), fuzzy systems (FSs), artificial immune systems (AISs), evolutionary computation (EC), and SI algorithms. NNs are defined as systems which can be trained; they are strong in pattern recognition. FSs are described as systems capable of reasoning under uncertainty. AISs are mainly applied for the recognition of patterns to perform classification tasks to cluster. EC algorithms are considered as a type of stochastic optimization search, and SI algorithms are considered a strong combinational optimization search (Engelbrecht, 2007).

Figure 4.1 illustrates the CI relationships. Probabilistic methods are usually used together with CI. Each of the CI models are based on biological systems. NNs models are inspired by biological neural systems, EC models are based on evolution (including genetic and behavioral evolution), SI models recreate social behavior of organisms in swarms or colonies, AIS models are based on the human immune system, and FS models started from analysis of organisms interacting with their environment (Engelbrecht, 2007).

![Figure 4.1. Computer intelligence paradigm.](image)
The most popular evolutionary algorithm (EA) which falls into the EC category is the Genetic Algorithm (GA). GAs emulate the natural evolutionary phenomenon, and each species keeps favorable adaptations in an environment which is constantly changing (Karaboga, 2007). Evolutionary algorithms generally start with an initial population of organisms called the initial solutions, then during execution, the algorithm allows the initial population of organisms to mutate and recombine. They select the fittest organisms to survive at each generation, resulting in a process of refining the solutions (Karaboga, 2014). GAs were first established by Holland (1992). The NSGA-II, which is a version of a GA, has been used for the automatic generation of traceability links (Ghannem et al., 2017; Ghannem et al., 2016).

In our research, we explore another intelligent algorithm to optimize and automatize RTR. We apply a SI algorithm, the ABC, and evaluate the effectiveness of its application to the RTR problem. The term swarm usually refers to any group of interacting agents or individuals. A broadly used example of a swarm is bees swarming around their hive; however, we can extend the metaphor to other systems having a similar architecture.

4.2.2. Swarm intelligence-based algorithms

Research in SI began in the early 1990s. SI algorithms reproduce the idea of swarms, which is often defined as optimization via emergence which refers to the existence of patterns, larger entities, or regularities. SI simulates the behavior of a group of self-organized and multi-agent systems present in nature or natural processes. The term swarm defines a collection of animals such as fishes, birds and insects including ants, termites and bees conducting collective behavior. In swarms, the individual agents behave without supervision. However, as a result of their
perception of the neighborhood, each of these agents has a stochastic behavior. SI is defined as “the collective behavior of decentralized and self-organized swarms” (Karaboga, 2014).

A population of agents in the SI category interacts locally, mostly following simple rules, and leads to an intelligent global behavior (Hanne & Dornberger, 2017). In SI, a single individual usually has minimum importance since the intelligent behavior takes place after interaction with other similar individuals. This swarm behavior is broadly used to find solutions to optimization and search problems. The area of logistics is a popular area of application for SI since often combinatorial aspects result in a large search space (Hanne & Dornberger, 2017). Stochastic mechanisms are used to work on the population in order to explore the search space and solution paths without requiring additional information about the problem.

Swarm Algorithms (SAs) are considered a sub-category of both SI and EC since they utilize the principles of evolution and of swarms. SAs, the most representative method of SI (Hanne & Dornberger, 2017), include Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Bees Algorithms (BAs), and Cuckoo Search (CS) (Hanne & Dornberger, 2017). In this research, we apply the ABC to the RTR problem. We use ABC to generate possible combinations of requirements (REQ) and source code classes (SCC) which represent traceability links (REQ, SCC). Through iterations, the algorithm finds the list of links that as a set maximizes the objective function.

4.2.3. Artificial Bee Colony (ABC)

ABC is one of the most recent SI algorithms; it was proposed by Karaboga (2005). ABC is inspired by the intelligent behavior of honeybees which find flowers, known as food sources, and they share the information related to the food sources with other bees in the nest. This
algorithm is simple and easy to implement (Karaboga & Akay, 2007). ABC has artificial agents (bees) which are classified into three types: the employed bee, the onlooker bee, and the scout bee. Each of these types of bees performs different jobs in the algorithm (Karaboga, 2005).

The employed bees concentrate on a food source and keep the position of that food source in their memories. Each employed bee is associated to exactly one food source. Thus, the number of employed bees and the number of food sources are the same (Awadallah et al., 2019). Each employed bee saves the information of the food source in its memory; every employed bee temporarily shares its information about the food source with the onlooker bee. Onlooker bee chooses the best food source from those found by the employed bee, and further explores for a new food source (generated by mutation) around the selected food source (Awadallah et al., 2019). Employed bee, whose food sources are abandoned by the onlooker bee because they cannot be further improved, automatically becomes a scout bee. Scout bee randomly explores the search space looking for new food sources and new nectars (Awadallah et al., 2019). The ABC algorithm, similar to other population-based algorithms, uses an iterative improvement process to find the best solution.

ABC uses a combination of local and global search methods. Employed bees and onlooker bees guide the local search while onlooker bees and scout bees manage the global search. Food sources (flowers) are the possible solutions, and the fitness value of the solution corresponds to the nectar of the associated food source according to the ABC representation (Alzaqebah & Abdullah, 2011).

The advantages of ABC include easy to implement, robust, and highly flexible. It is defined as highly flexible because it only requires two control parameters, the maximum cycle number and
the colony or solution size (Karaboga, 2007). In addition, bees can be added or removed without having to reinitialize the algorithm (Karaboga, 2007). ABC can be applied to several optimization problems without requiring any modification, and compared to other search techniques, it uses fewer control parameters (Karaboga, 2007). The disadvantages of ABC are that it is slow when used in serial processing, and it needs a large number of objective function evaluations (Karaboga & Akay, 2009; Abu-Mouti & El-Hawary, 2012; Ab Wahab et al., 2015).

Several fields have benefited from the use of ABC, including engineering design problems (Gerhardt & Gomes, 2012; Sharma & Pant, 2012), networking (Lee et al., 2012), business (Lee et al., 2012), electronics (Karaboga, 2014), scheduling (Karaboga, 2014) and image processing (Karaboga, 2014), demonstrating the popularity of the approach. The ABC has been modified a number of times (Khader et al., 2013). Bao and Zeng presented three approaches which are used for onlooker bees for selecting food sources in ABC. They are the Rank Selection Strategies ABC (RABC), Tournament Selection ABC (TABC) and Disruptive Selection ABC (DABC) (Bao & Zeng, 2009). The variants aim to expand the population diversity and avoid premature convergence (Bao & Zeng, 2009; Alzaqebah & Abdullah, 2011). In this research, we apply the original or standard implementation of the ABC.

4.3. Related work

Our approach using the ABC algorithm for the RTR problem is controlled by an IR-based objective function which is the mathematical representation of the problem that needs to be optimized. We defined our objective function by applying IR techniques to establish relationships among requirements and source code according to their textual content. In this section we present
an overview of IR techniques applied to the RTR problem and explore existing research which applied optimization techniques to the RTR problem.

4.3.1. IR techniques applied to the RTR problem

Software development carries many challenges for software developers; one of these challenges is maintaining traceability links across software artifacts. Traceability links are important due to numerous benefits for the overall quality of the software, but maintaining them is a complex task since the manual recovery of these links is time consuming and tedious. Several IR-based approaches and tools have been proposed to help solve the RTR problem. IR techniques applied to RTR seek to find possible traceability links among requirements and other artifacts by using their textual content.

An approach which is applied to object-oriented source code, such as C++, to create links among source code classes and instruction pages (manual) sections is presented in (Antoniol et al., 1999). This research suggests the generation of a language model from each section of the manual and the transformation of the text of the source code to generate a useful representation that minimizes some coding styles problems such as mnemonic concatenation and abbreviations. Then, the language model and the textual representation of the source code are compared to match instructions pages with segments of code.

Application of IR probabilistic is found in (Antoniol et al., 2000). In this research IR probabilistic is applied to object-oriented source code to find traceability links across functional requirements and source code classes. Results validated over the Albergate data set, a computational system for the management of small/medium sized hotels, show an average value of 50% Precision (P) and recall (R).
The application of classical VSM to improve the candidate link generation is presented in (Hayes, 2003). Authors claimed that "this research in comparison to other existing tools and approaches did not generate better results of P and R, but the approach is faster and required less information".

A research effort that compares different IR methods for the recovery of requirement traceability links is presented in (Mahmoud & Niu, 2014). Studied methods include VSM with thesaurus support (VSM-T), Part-of-Speech-enabled VSM (VSM-POS), LSI, latent Dirichlet allocation (LDA), explicit semantic analysis (ESA), and normalized Google distance (NGD) using three different datasets. Methods were compared to the baseline approach VSM. Experimental results show that VSM with domain thesaurus (VSM-T-TD) achieved a comparable performance to the baseline approach, and it is selected as the most reliable method among the studied methods. Additionally, the research states that the baseline approach outperformed the majority of the studied methods, supporting the application of VSM to model the objective function in our intelligent approach.

Another research effort that studies different IR techniques applied to RTR compares their effectiveness; it studies the application of IR probabilistic, VSM and LSI for creating traceability links among requirements and source code classes (Rodriguez & Carver, 2019). The approach is applied to object-oriented source code, and results are validated over two public data sets based on P and R. Results over the two data sets suggest high P above 90% when using IR probabilistic and VSM techniques, and low R. Additionally, the approach returned low P and R when evaluated using LSI (Rodriguez & Carver, 2019).
IR-based tools are available for application to the RTR problem. RETRO, proposed by Hayes et al. (2007), is a semi-automatic tool that implements the IR methods of VSM and LSI to generate a requirements traceability matrix (RTM). ADAMS Re-Trace, proposed by De Lucia et al. (2005), is a tool that implements LSI for identifying traceability links across different artifacts of the system. Trustace presented by Ali et al. (2011) combines IR techniques and a data mining approach to create traceability links across the requirements and the sources code.

However, most of the approaches and tools described are time consuming or semi-automatic. To improve the RTR and to achieve automatization in the process, we developed an approach that combines IR techniques with an optimization algorithm (ABC). Our approach is motivated by the high P results returned by some of the IR approaches, the popularity of IR applied to RTR, and effectiveness of optimization algorithms.

4.3.2. Traceability link recovery as an optimization problem

Regardless of the benefits, stakeholders frequently do not enforce RTR. Often traceability links are nonexistent or not updated. The need for an automatic and accurate approach for the generation of traceability links is of interest for a number of researchers who are aware of the benefits for the quality of software and who understand that due to factors such as time, number of people on the team or miscommunication, traceability links often are not created or updated (Pennington, 1987; Pinheiro & Goguen, 1996; Arnold & Bohner, 1993). Some researchers have attacked automatization of RTR by applying optimization algorithms. Genetic algorithms have been applied to automate RTR in previous research (Ghannem et al., 2017; Ghannem et al., 2016).

A solution based on a mono-objective genetic algorithm for matching a requirement with one or many classes of the source code by comparing textual elements of the source code such as
identifiers, API documentation, and comments with a textual description of requirements is presented in (Ghannem et al., 2016). The approach achieved average P of 86% and an average R of 83%.

Additionally, application of a multi-objective approach based on the NSGA-II which uses three (3) objective functions based on semantic similarity and information of frequency of changes is described in by Ghannem et al. (2017). Results were above 80% for P and R, supporting the application of an optimization algorithm in the traceability field. However, “if no information or only incomplete information is available about the times of change of the artifacts”, the approach will not be able to take advantage of the recency of change and frequency of change objectives and optimize them” (Ghannem et al., 2017).

Another approach using the NSGA-II is presented in chapter 3 of this manuscript. The generation of a mathematical representation based on the frequency of changes needs extra information that often is not provided or available for the recovery of traceability links. A multi-objective approach based on different IR techniques to establish textual similarities among requirements and source code classes is described in chapter 3. This research uses as input the requirements document and the source code, and applies IR methods for finding similarities in the textual content of requirements and source code. Objective functions are based on two popular methods to establish text similarity, Jaccard similarity and weighted cosine similarity. The assumption is that a strong similarity among the textual content of the requirements and the source code classes suggests a possible link. By maximizing the objective functions, the links from requirements to source code with the highest similarity values are identified. Results support the potential of applying optimization algorithms combined with IR techniques to address the RTR problem. The approach, which was tested over three public data sets (EBT, Albergate and eTour),
returned values for P and R above 80% on average. It is an alternative when minimal information about the system is available.

Most of the works described in this section rely on the application of genetic algorithms. In this study we propose the application of a swarm intelligence approach which is a different optimization algorithm. Our research is motivated by the positive results obtained by previous research that applied optimization algorithms to RTR. However, in order to find improvements and better alternatives, we present another approach that relies on the application of a single objective function, the ABC algorithm, categorized as a SI algorithm. ABC is considered to be simpler and have faster convergence rates when compared to GAs (Karaboga, 2005).

4.4. **ABC algorithm**

This section presents an overview of ABC. The ABC consists of four main steps: initialization, send employed bee, send onlooker bee, and send scout bee. After the initialization step, the other three main steps of the algorithm are iteratively executed stopping when a defined termination condition is met. Figure 4.2 illustrates the process.

![Figure 4.2. ABC algorithm process.](image-url)
**Step I: Initialize the parameters of ABC.** Most of the optimization algorithms share this same initial step. For ABC, parameters include *Solution Number (SN)*, *Maximum Cycle Number (MCN)* and *limit*. SN is associated with the number of food sources in the population, and it can be compared to the population size parameter of genetic algorithms. MCN controls the maximum number of generations, and *limit* is the number of tries for improvements before the abandonment of the food source.

**Step II: Generate the initial population.** The ABC starts with generating a population of initial solutions (food sources), \( \text{population} = [X_1, X_2, \ldots, X_{SN}] \) of size Solution Number (SN). The generation of each solution is a random process given by Eq. (4.1):

\[
X_{s,j} = LB_j + (UB_j - LB_j) \times U(0,1)
\]  

(4.1)

where \( s \in (1, 2, \ldots, SN) \), \( j \in (1, 2, \ldots, D) \), \( UB_j, LB_j \) are the upper and lower bounds for the decision variable \( X_{(s,j)} \), \( D \) is the length of each solution, and the value of \( U(0,1) \) is a uniform random number between 0 and 1.

**Step III: Send the Employed bee.** Every food source is related to exactly one employed bee. In the employed bee phase, each employed bee visits a food source and modifies the food source \( x_s \) in its memory, producing a new neighboring food source \( x'_s \) using search Eq. (4.2). New candidate solutions are produced for each employed bee by crossover and mutation of the employees. A greedy selection is applied between \( x_s \) and \( x'_s \) for selecting the solution with the higher associated value (nectar). When the nectar value of the new food source \( x'_s \) is higher than the nectar value of the old food source \( x_s \), the employed bee memorizes the new food source; otherwise, it keeps information for the old food source. The search equation is defined as:
\[ x'_{s,j} = x_{s,j} + \text{rand} (-1,1) * (x_{s,j} - x_{k,j}) \] (4.2)

where \( x_{k,j} \) represents the variable \( x \) at the \( J_{th} \) position in a food source \( x_k \). The value of \( x_{k,j} \) is chosen randomly by an employed bee. It has to be different from the original food source \( x_s \). The value of \( \text{rand} (-1,1) \) is a random number ranging between \(-1 \) and \( 1 \) that controls the generation of a neighboring food source around \( x_{s,j} \), which will be compared to the original food source.

**Step IV: Calculate the probability values.** After the employed bee completes its search, it shares the food source information with the onlooker bee. The onlooker bee calculates the nectar of all the food sources taken from the employed bee and selects a food source with a probability related to its nectar quality. Our implementation of the original ABC applies the proportional selection scheme for calculating the probability values, given by Eq. (4.3).

\[
\text{Probability}_s = \frac{f_s}{\sum_{i=1}^{SN} f_i} \quad (4.3)
\]

where \( f_s \) represents the fitness cost (nectar) of the food source \( x_s \) in position \( s \), and \( \text{Probability}_s \) is the probability of the food source \( x_s \) which is determined by the nectar quality of that food source.

**Step V: Send the onlooker bee.** Unlike the employed bee, onlooker bee selects a food source based on the \( \text{Probability} \). During the onlooker phase, the onlooker bee evaluates the nectar of all the food sources determined from the employed bee. The onlooker bee chooses a food source \( x_s \) according to the value of \( \text{Probability}_s \). A new neighboring food source \( x'_s \) is created by using the same search Eq. (4.2) applied to the selected food source \( x_s \). As it is performed in the employed
bee phase, a greedy selection is carried out between $x_s$ and $x'_s$. If the new neighboring food source $x'_s$ has a higher nectar value compared to the old one $x_s$, then $x'_s$ will replace the old food source in the onlooker bee’s memory and $x'_s$ becomes part of the population.

Onlooker bee performs a selection process once it gathers the information from the employed bees. The selection strategy is important for the algorithm. In our implementation, we use the proportionate selection, also known as roulette wheel selection, used for the original algorithm. In a roulette wheel selection, the circular wheel is divided according to range of values of the objective function. The fitter individual has a greater portion on the wheel and it results in a greater possibility of been selected when the wheel is rotated. Thus, the probability of choosing an individual depends directly on its fitness. By using this type of roulette wheel based probabilistic selection, we ensure that better food sources are more likely to be visited by an onlooker bee. Therefore, the onlooker bee discovers new candidate food sources surrounding good solutions.

*Step VI: Send the scout bee.* In case the food source $x_s$ cannot be improved after a predefined number of cycles (determined by limit), the onlooker bee abandons the food source. Then, the employed bee associated to the food source automatically becomes a scout bee. The scout bee randomly explores the search space to generate a new food source using Eq. (4.1).

*Step VII: Stop condition.* Steps III − VI are repeated until the stop parameter $MCN$ is met. $MCN$ is randomly assigned to achieve convergence.

4.5. **Modified ABC to solve the RTR problem**

We modified and adapted ABC to the RTR problem. The first step of ABC is to generate the initial population. In our implementation, the population is not represented by a single decimal
number as in the original ABC; instead, it is represented by integer vectors formed by a set of pairs from requirements and source code classes which are randomly generated and regulated by a parameter that we introduced and defined as the maximum size, called $Max\text{-size-sol}$, which is the maximum number of pairs composing a food source vector.

The input of our implementation is requirements and the matching source code (classes) and pair combination of them represent the food sources. A pair contains a unique number $ID$ of exactly one requirement from the requirements document and a class from the system source code, denoted as REQ and SCC, respectively. $Max\text{-size-sol}$ depends on the number of REQs and SCCs in the data set. As a result, food sources can have different sizes depending on the data set. In that sense, data structures and internal implementation of the algorithm were modified to adjust to the changes.

The pairs conforming to a food source are integers related to the requirements $ID$ and the source code $ID$ in the data sets. For the random generation of food sources, we use lower and upper bounds. The lower and the upper bounds, which are used in the original algorithm to define a range for the random generation of the decimal values, in our implementation are used to ensure that the random generation of values associated to requirements $IDs$ and source code $IDs$ produces existent $IDs$. The lower bound was assigned to zero, which the minimal unique $ID$ for requirements and source code and the upper bound was related to the maximum number of requirements $IDs$ and Source Code $IDs$ in the data set. We generated the numbers in the vector that form the food sources as integers within the lower and upper bounds with a random integer function. As a result, we maintained the combinational factor in the algorithm.
Each vector is treated as a single solution (food source) associated to a single employed bee and having a unique value for the objective function (nectar). In ABC, the position of a food source is a possible solution to the optimization problem, and the nectar value of a food source corresponds to the quality (objective function value) of the associated solution. These relationships are shown in figure 4.3.

![Figure 4.3. Terminology mapping of RTR to ABC algorithm.](image)

We randomly generated an initial population formed by food sources and regulated by the parameter $SN$, which is the maximum number of food sources in the population. $SN$ controls the number of food sources evaluated at each iteration and controls the number of employed bees generated. We created a population of candidate solutions that evolves towards the best solution. The goal is to find a food source that maximizes, as much as possible, the semantic similarity based on weighted cosine similarity using $TF-idf$ across pairs (REQ, SCC).

At each iteration, if the $MCN$ is not met, we evaluate the food sources based on the objective function (nectar). Each vector solution formed by the pairs of REQ, SCC represents a unique food source, and the evaluation of the objective function was based on calculating the mean of the vector solutions after evaluating each individual pair, in contrast to the original implementation of ABC where each food source is associated to a single decimal value.
Next, we generate new solutions using mutation for the employed bee and evaluate again the solutions based on the objective function. We apply a greedy selection process for employed bee. Then, we calculate the probability values for the current solution so that the onlooker bee can choose one (the best) according to its value. We assign the onlooker bee to the solutions according to the probabilities, generate new solutions using mutation, evaluate the objective function, and apply the greedy selection process for the onlooker bees. The mutation operator have been modified to adapt to new data structures; detailed information is provided in subsection 4.5.3.

An employed bee abandoning a solution means that after a number of tries, defined by the parameter limit, the solution could not be improved. In that case, the bee stops its exploitation, it turns into a scout bee, and the food source is replaced with a new solution for further exploration.

At each iteration, the bees memorize the best solution found and check the termination criterion MCN. If MCN is not satisfied, the algorithm iterates; otherwise, the iteration stops.

The mutation operator and the data structures used for the algorithm were modified to adapt to the RTR problem. The original mutation operator is based on applying mathematical operations to a food source to generate a new food source within the lower and upper bounds for evaluation. In our implementation, our data has additional restrictions. We use a vector instead of an single decimal value to represent our food source; additionally, application of mathematical operators to each value of our food source could produce decimal results which will break the execution since the new generated value, even if it is within the lower and upper bounds, will not be associated to any requirement or source code in the data set. Our mutation operator needs to generate integer values in the defined range (lower and upper bounds) to ensure that the new value corresponds to an available ID which allows the text analysis.
4.5.1. Food source representation

We represent solutions (food source) using a list of (REQ, SCC) pairs where REQ denotes the ID of the requirements and SCC denotes the ID of the source code class, represented by numerical values. Table 4.1 shows an example of this representation.

<table>
<thead>
<tr>
<th>REQ</th>
<th>SCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ115</td>
<td>SCC46</td>
</tr>
<tr>
<td>REQ117</td>
<td>SCC48</td>
</tr>
<tr>
<td>REQ121</td>
<td>SCC95</td>
</tr>
<tr>
<td>REQ121</td>
<td>SCC93</td>
</tr>
</tbody>
</table>

4.5.2. Objective function for evaluation of food sources

Our implementation is based on textual similarity among REqs and SCCs. Requirements and source code classes are free text documents holding valuable semantic information. By extracting key words from the REQ, and comparing those queries to SCC identifiers such as variables, classes and methods names, which in this research are identified as the elements that hold the most semantic information about the code, we can establish connections among a REQ and a SCC. A strong similarity across a REQ and a SCC is assumed as a possible traceability link between those two elements. As a result, we based our objective function, which models the execution of the optimization, as a weighted cosine similarity, which is a popular method for finding textual similarity. Weighted cosine similarity is calculated by applying frequency of term and the inverse document frequency factor (TF-idf) to create weighted vectors which are evaluated using cosine similarity to measure the similarity of the textual content of every pair (REQ, SCC) in the food source.
The goal is to maximize the objective function (SemSTF{idf/CS}). The objective function
SemSTF-idf/CS value associated with the semantic similarity for a food source (nectar) is the
average of the semantic similarity values of each pair (REQi, SCCi) in the food source.
SemSTF-idf/CS is given by Eq. (4.4), where m is the number of pairs in the food source.

\[
TF_{(t,d)} = \frac{\text{Number of times term (t) appears in the document (d)}}{\text{Total number of terms in the document (d)}}
\]

\[
idf_{(t,D)} = \log \frac{(\text{Total number of documents (D)})}{(\text{Number of documents with term (t) in it})}
\]

\[
TFidf_{(t,d,D)} = TF_{(t,d)} \cdot idf_{(t,D)}
\]

\[
SemSimTF(I) = \sum_{i=1}^{n} \text{CosineSimilarity}(\text{Weight}_{TF-idf}(REQ_i, SCC_i))
\]

The set that maximizes the objective function is returned as the optimal solution.

4.5.3. Mutation

During the employed and onlooker phases of ABC, we apply a mutation operator to food
sources associated to the bees to generate new solutions and establish a comparison aiming to
improve the objective function value. Once we generate a new food source, we compare the new
solution with the old solution, and the value of the bee is updated to the one that holds the best
value for the objective function (best nectar).

The mutation operator that we implemented differs for the original ABC mutation operator
since the original mutation is based on numerical decimal values and for our particular problem
we need to mutate values to generate integers in a particular range associated to the values \text{IDs} in
the requirements document and the source code. Our mutation operation needs to keep the new
values inside the lower and upper bounds and values need to maintain certain properties (integers).
A mutation based on random generation of decimal number such as the one implemented in original algorithm is not feasible for our implementation.

Our mutation operator randomly changes pairs of the food source according to a randomly determined mutation factor which determines if we mutate the pair. Once we select a bee related to a specific food source, the mutation operator randomly selects three pairs (REQ, SCC) of the food source and replace the pairs with other different pairs which are created by applying Eq. (4.5). The new values are the result of adding the randomly selected pair from the current food source to a random number and subtracting this result from the pair in the same position of another randomly selected food source. We derived Eq. (4.5) by experimentation to allow for diversity in the search space.

\[
\text{MutatedPair} = \text{CurrentFoodSource}[X] + (\text{randomnum} \times 2) - (\text{AnotherFoodSource}[x] - \text{CurrentFoodSource}[X]) \tag{4.5}
\]

where X is a randomly defined pair. The random number was defined to 30 for the EBT and Albergate, and 50 for the eTour after experimentation.

After we perform the mutation, we call another function which checks that generated numbers are in the predefined bounds. Bounds, defined by the number of REQs and SCCs in the data set, range from 0 to the maximum ID number for REQs and SCCs. In the checking function, we evaluate each of the three selected pairs to mutate. If the resulting values of REQ and SCC are less than zero, the mutated values are set to zero which is the lower bound. In contrast, if the generated values for REQ and SCC are greater than the defined upper bounds, we assign values chosen randomly from the solution space. Figure 4.4 illustrates the effect of the mutation operation.
4.6. **ABC implementation and experiment setup**

4.6.1. **ABC implementation for the RTR problem**

We implemented our adaptation of ABC using Python 3.6. The algorithm has two phases. The first phase transforms the raw input data from requirement documents and source code into representations for use with ABC. Figure 4.5 illustrates the preprocessing which removes stop words, punctuation and capitalization of words. The source code preprocessing keeps only the identifiers names, including variables, methods, and classes. We manually pre-processed the input data to prepare them for the application of the ABC.

![Figure 4.5. Preprocessing phase.](image)
The second phase is the actual implementation of ABC. The ABC aims to maximize the objective function. It iterates until $MCN$ is reached. Once the algorithm stops, it outputs the population that maximizes the objective function, known as the best food source. Figure 4.6 illustrates the process.

**Figure 4.6.** ABC algorithm adapted to the RTR problem.
4.6.2. Experimental setup

We use three open-source projects, Event Based Traceability (EBT), Albergate, and Tour Guide System (eTour), to test our application of ABC to RTR. All data sets were downloaded from http://coest.org/ (CoEST). The CoEST is a website created to provide data sets for research or RTR investigation. We selected EBT, Albergate and eTour since they represent small, medium and large sized data sets that include the requirements document, source code divided by classes, and traceability links from the requirements document to the classes of the system.

The EBT data set consists of 40 requirements, 50 Java source code classes, and 98 trace links from requirements to classes (Cleland-Huang et al., 2003). Albergate has 17 requirements, 55 source code classes, and 54 traceability links from requirements to classes (Antoniol et al., 2000). The eTour data set includes 58 use cases used to derive the requirements, 116 Java source code classes, and 308 trace links from requirements to classes (Poshyvanyk et al., 2011). We used the traceability links in the data set to validate our findings.

We defined parameters $SN$ and $MCN$ based on experimentation to achieve convergence. Parameters $SN$ and MCN were set to 200 and 200 respectively for EBT, Albergate and eTour. The ABC is an effective exploratory algorithm with a fast convergence rate. In that sense, we did not need high values for $SN$ and $MCN$ for achieving convergence. We defined the parameter $max-size-sol$ based on experimentation to 250 for EBT, 150 for Albergate and 400 for eTour. The $limit$ parameter is an arbitrary number defined based on experimentation. $Limit$ was set to 50 for better results; solutions abandoned too early or too late could negatively affect the exploratory process that leads the algorithm to converge to a global maximum solution. The employed bee which cannot improve its solution until it reaches the limit of trials becomes a scout bee. Therefore, scout
bees in the ABC algorithm prevent inactivity of the employed bees population. Intuitively, this method provides an easy means to overcome any local optimal solution in which a bee may have been trapped.

Our output shows the list (REQs, SCCs) of traceability links, and it illustrates the evolution of the objective function which reaches the maximum value at the last iteration. Additionally, in the output we show the value for the best solution which is directly related to the number of matches found. Figure 4.7 illustrates an output example for the EBT data set.

![Figure 4.7. ABC output.](image)

4.7. **Experimental results**

4.7.1. **Can the SI algorithm ABC and an IR-based objective function assess automatic RTR?**

We evaluated our results based on the IR metrics of P and R. P is the division of number of the correct links detected by the total number of links detected. R is the ratio of the number of links detected to the number of correct links. P represents the correctness of the results while R
denotes the completeness of the results. High values of P and R are desired. P and R are described by Eq. (4.6) and Eq. (4.7) respectively.

\[
P = \frac{TP}{TP+FP} \in [0,1]
\]  

(4.6)

\[
R = \frac{TP}{TP+FN} \in [0,1]
\]  

(4.7)

To calculate P and R, we identified for each returned local optimal solution, values of true-positive values (TP), false positive values (FP), and false negative values (FN). TPs are the correct pairs in the output, FPs are pairs that appear in the output but are incorrect, and FNs are pairs that should be in the output but are missing.

There is a random factor in the generation of the initial population formed by the initial combination of requirement (REQ) and source code (SCC). As a result, the output oscillates at each execution. For more accurate results, we iteratively executed the algorithm 10 times for each data set, and we calculated the mean of P and R based on the iterations (i), computed by Eq. (4.8) and Eq. (4.9).

\[
MeanP = \frac{\sum_{i=0}^{10} P_i}{10}
\]  

(4.8)

\[
MeanR = \frac{\sum_{i=0}^{10} R_i}{10}
\]  

(4.9)

Table 4.2 shows P, R, the number of correct links, and total number of links found for each of the 10 iterations for the EBT data set, along with its respective mean values. The experimental results show that both P and R values range above 70%. Results indicate that the implementation of ABC tested over EBT found a mean of approximately 100 links out of the 98 defined in the
original data set representing an ideal value since it reduces the number of FPs and FNs to their minimal value, increasing P and R.

Table 4.2. P and R for 10 iterations over EBT data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73</td>
<td>100</td>
<td>73%</td>
<td>74%</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>101</td>
<td>69%</td>
<td>71%</td>
</tr>
<tr>
<td>3</td>
<td>72</td>
<td>104</td>
<td>69%</td>
<td>73%</td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td>103</td>
<td>73%</td>
<td>77%</td>
</tr>
<tr>
<td>5</td>
<td>68</td>
<td>97</td>
<td>70%</td>
<td>69%</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>99</td>
<td>70%</td>
<td>71%</td>
</tr>
<tr>
<td>7</td>
<td>74</td>
<td>101</td>
<td>73%</td>
<td>76%</td>
</tr>
<tr>
<td>8</td>
<td>69</td>
<td>99</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>9</td>
<td>71</td>
<td>98</td>
<td>72%</td>
<td>72%</td>
</tr>
<tr>
<td>10</td>
<td>69</td>
<td>99</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>Mean</td>
<td>71.1</td>
<td>100.1</td>
<td>70.9%</td>
<td>72.3%</td>
</tr>
</tbody>
</table>

Figure 4.8 shows a graphical representation of the results obtained for P and R at each iteration of the EBT data set. The best results for P occurred in iterations 1, 4 and 7 with a 73% P. Iteration 1 retrieved 73 correct links out of the 100 links found and iteration 4 retrieved 75 correct links out of the 103 links found. Regarding R, the best value occurred in iteration 4 which returned 77% R. Iteration 4 returned the most complete list of links recovering 75 links out of the 98 traceability links in the original data set while generating 23 FNs. The difference between the average value of P and R was low. However, we observed that R is slightly better, which translates to results more complete than accurate. Our approach generated more FPs than FNs in general.
Table 4.3 shows P and R for each of the 10 iterations for the Albergate data set. Similar to EBT, it also shows the number of correct links found, total number of links found and its respective mean values. The experimental results over Albergate indicate that both P and R values range above 75% on average. Albergate returned lower values of P and R in comparison to the EBT, but still results are considered to be in a good range. For Albergate, our approach returned an average of 43 correct links out of an average of 58 total links; however, the number of correct links in the original data set is 54. Thus, for Albergate, our approach introduced more FNs and FPs to the results in comparison to EBT, affecting the values of P and R.

Figure 4.9 shows a graphical representation of the results obtained for P and R at each iteration of the Albergate data set. The best result for P was found in iteration 10 with a value of 78%, which returned 47 correct links out of the 60 links found. Regarding R, the best value was found in iteration 3 which returned 46 links out of the 54 links in the original data set, representing an 83% R. For Albergate, in most of the iterations the value of R was higher than the value of P indicating that the list of links returned is more complete than precise. Most of the links were found correctly, but FPs were introduced.
Table 4.3. P and R for 10 iterations over Albergate data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>55</td>
<td>73%</td>
<td>74%</td>
</tr>
<tr>
<td>2</td>
<td>43</td>
<td>56</td>
<td>77%</td>
<td>80%</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>60</td>
<td>77%</td>
<td>85%</td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>60</td>
<td>75%</td>
<td>83%</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>56</td>
<td>71%</td>
<td>74%</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>61</td>
<td>69%</td>
<td>78%</td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td>55</td>
<td>69%</td>
<td>70%</td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>56</td>
<td>75%</td>
<td>78%</td>
</tr>
<tr>
<td>9</td>
<td>43</td>
<td>61</td>
<td>70%</td>
<td>80%</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
<td>60</td>
<td>78%</td>
<td>87%</td>
</tr>
<tr>
<td>Mean</td>
<td>42.6</td>
<td>58.0</td>
<td>73.4%</td>
<td>78.9%</td>
</tr>
</tbody>
</table>

Figure 4.9. Values of P and R for 10 iterations over the Albergate data set.

Table 4.4 shows the values of P and R for 10 iterations over the eTour data set. As in previous data sets, it shows the total number of correct links, total number of links retrieved, P, R and their mean values. Similar to the EBT and Albergate data sets, for eTour, which is a larger data set, both P and R had high values that range above 70%. We found a mean of 308 links which
is the exact number of links defined in the original data set. In contrast to Albergate in which the results show that in most of the cases the values for R were higher than the P values, eTour and EBT which are larger data sets in comparison to Albergate, have similar behavior. For eTour and EBT, results fluctuate without following a defined pattern. Some iterations returned higher values of P and some returned higher values of R. Additionally the number of total links recovered for both data sets EBT and eTour is similar to the number of links in original data sets.

Table 4.4. P and R for 10 iterations over eTour data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>233</td>
<td>315</td>
<td>74%</td>
<td>75%</td>
</tr>
<tr>
<td>2</td>
<td>238</td>
<td>318</td>
<td>75%</td>
<td>77%</td>
</tr>
<tr>
<td>3</td>
<td>225</td>
<td>300</td>
<td>75%</td>
<td>75%</td>
</tr>
<tr>
<td>4</td>
<td>223</td>
<td>305</td>
<td>73%</td>
<td>72%</td>
</tr>
<tr>
<td>5</td>
<td>232</td>
<td>310</td>
<td>75%</td>
<td>75%</td>
</tr>
<tr>
<td>6</td>
<td>235</td>
<td>312</td>
<td>75%</td>
<td>76%</td>
</tr>
<tr>
<td>7</td>
<td>229</td>
<td>301</td>
<td>76%</td>
<td>74%</td>
</tr>
<tr>
<td>8</td>
<td>233</td>
<td>316</td>
<td>74%</td>
<td>76%</td>
</tr>
<tr>
<td>9</td>
<td>236</td>
<td>298</td>
<td>79%</td>
<td>76%</td>
</tr>
<tr>
<td>10</td>
<td>213</td>
<td>303</td>
<td>70%</td>
<td>69%</td>
</tr>
<tr>
<td>Mean</td>
<td>229.7</td>
<td>307.8</td>
<td>74.6%</td>
<td>74.5%</td>
</tr>
</tbody>
</table>

Figure 4.10 illustrates the graphical representation of the values for P and R at each iteration over the eTour data set. We observed a constant difference ranging from 2% and 3% between P and R in the majority of the iterations, and values for P and R above 70% at all iterations. The best precision value was found in the 9th iteration with a P value of 79% returning 236 correct links out of the 298 retrieved, and the best performance of R is in the 2nd iteration where R was 77%
returning 238 correct links out of the 308 in the original data set. At its best iterations, our implementation generated 72 FPs and 70 FNs links.

![Figure 4.10. Values of P and R for 10 iterations over the eTour data set.](image)

Table 4.5 shows the mean values of P and R over EBT, Albergate and eTour data sets. For EBT, the mean P was approximately 70%, and the mean R approximately 72%. For Albergate, the mean P was approximately 73%, and the mean R was approximately 79%. For eTour, the mean P was approximately 75%, and the mean R was approximately 75%. We observed that Albergate was the data set with the highest gap between mean values of P and R in comparison to EBT and eTour; in the majority of the iterations Albergate had higher values of R than P. Additionally the mean of total of links retrieved for Albergate was higher than the value in the original data set, which translates to an increase of the number of FPs associated to P values. In contrast, the performances of EBT and eTour were similar. High P and R values were found for both data sets, and results did not follow any specific pattern. However, in terms of P and R, our approach performed better for eTour which is a larger-sized data set. In both data sets, EBT and eTour, the number of total links retrieved was similar to the number in the original data set. As a result, values for P and R were similar.
Table 4.5. Mean P and R for 10 iterations over EBT, Albergate and eTour data sets.

<table>
<thead>
<tr>
<th></th>
<th>EBT</th>
<th>Albergate</th>
<th>eTour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean P</td>
<td>71%</td>
<td>73%</td>
<td>75%</td>
</tr>
<tr>
<td>Mean R</td>
<td>72%</td>
<td>79%</td>
<td>75%</td>
</tr>
</tbody>
</table>

We hypothesize that values of P and R could be further improved by changing experiment settings. Values such as Max-size-sol, solution number (SN), maximum cycle number (MCN), and mutation could be modified with more experimentation. Modification of these parameters could bring more diversity to the search space which translates to a deeper exploration of the bees for finding better nectar sources.

Figure 4.11 shows the overall performance over EBT, Albergate and eTour data sets based on the mean values of P and R. We identified higher values of P in the eTour data set. In eTour, we recovered a mean of 230 links out of the 308 links shown in the original data set. However, results are also high for Albergate and EBT. Regarding recall the evaluation over Albergate returned the higher values with a mean of 79%, representing a mean of 43 correct links out of the mean 58 found.
Results for the adapted ABC are positive for applying an intelligent approach to the RTR problem to achieve automatization. The answer to RQ4-1 is satisfactory. The application of a swarm intelligence-based algorithm to the RTR produced encouraging results in terms of P and R. It helps to enable automatic RTR by building sets of links formed by a combination of different (REQ, SCC) pairs at each iteration which are a result of an exhaustive search that emulates how the hive bees’ behave in nature. At every iteration, a set of bees finds the best nectars which are the food sources or candidate solutions that maximize the objective function (local maximums). This research complements IR approaches by adding the search intelligence component that finds the best candidates link sets at each iteration.

4.7.2. **How effective is the application of CI algorithm to the RTR problem in contrast to IR techniques?**

In order to address RQ4-2, we compared our results to those results obtained by Rodriguez & Carver (2019) which use IR probabilistic, VSM and LSI techniques adapted to the RTR process. Figure 4.12 shows those results for P and R for EBT and eTour data sets, respectively. The experimental results, when comparing our approach to a baseline method, indicated that our approach outperformed the implementation of the IR techniques for R values on the two data sets. Although our implementation of ABC achieved slightly lower P values, results are compensated by R values.
Based on the assumption that both metrics P and R are equally important to define the efficiency of one technique over another, we computed the F1 score which is a weighted average of the P and R calculated according to Eq. (4.10).

\[
F1 = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]  

(4.10)

Figure 4.13 is a graphical representation of the F1 results. When comparing the techniques based on a balance from P and R values, our intelligent approach found an improved balance between P and R. F1 values for our approach based on ABC were higher in comparison with the values obtained for the basic implementation of IR techniques.

Additionally, our implementation of ABC aims to reduce the human intervention in the RTR process. The goal is to generate an alternative for the automatic generation of traceability links by using only a requirements document and the related source code as input. ABC applied to the RTR problem allows working with a big search space formed by a population of links. It automatically generated different lists of pairs (REQ, SCC), known as food sources, which were iteratively evaluated and modified to maximize the objective function. The resultant list of (REQ,
SCC) pairs was a complete list with the highest number of positive matches. In contrast, the implementation of the IR techniques (Rodriguez & Carver, 2019) only found for each requirement REQ the class SCC with the highest probability of being related with the specific requirement. In (Rodriguez & Carver, 2019) a threshold value was not assigned, and as a result, the implementation did not manage cases where one requirement was related to multiple classes, finding only a single pair for each requirement, affecting the R value associated to completeness.

![Figure 4.13. F1 values of ABC and IR techniques.](image)

**4.7.3. How effective is the application of a SI algorithm to the RTR problem in contrast to genetic algorithms?**

In order to answer RQ4-3, we compared our results using ABC to results obtained by Ghannem *et al.* (2017) which was based on an implementation of the NSGA-II algorithm adapted to RTR. In (Ghannem *et al.*, 2017), eTour and Albergate, two of the data sets used for our analysis, were used to validate findings. Figure 4.14 shows the comparison.
Results indicate that our implementation of the IR based ABC had lower P and R than the implementation of the NSGA-II presented by Ghannem et al. (2017) for both, eTour and Albergate. Regarding P, for Albergate our approach was less precise than the NSGA-II implementation by 10% and for eTour our approach was less precise that the NSGA-II implementation 16%. For R values our approach was only 1% less complete than the NSGA-II implementation for Albergate and 12% less complete for eTour. However, values obtained by both implementations are in good ranges. The main contribution is that it enables working only with the requirements document and the source code of the system, and by using minimal information from the system it keeps results comparable to the approach presented in (Ghannem et al., 2017) which is based on three objective functions and requires, in addition to the requirements document and the source code, information about chances history and frequency of changes.

4.8. Threats to validity

Because our adapted IR based ABC approach is mainly focused on the textual analysis of requirements and source code classes, if no information or only incomplete information is available about the requirements or the source code, we will not be able to maximize the objectives
and optimize them since that is the only input required. This restriction may affect the accuracy of the results.

Important assumptions for this research are that the requirements document does not contain abbreviations and the source code was developed applying good programming techniques, such as the utilization of meaningful names for identifiers (classes, methods and variables) enabling understanding of the code and the possibility of associating code to requirements. Lack of meaningful names or a good programming technique could represent a problem for the functionality of our approach since it is based on a semantic analysis; it would be difficult to establish relationships between the documents.

We assume that the structure of the requirements document allows identifying a clear list of individual requirements associated with a unique ID, and that the source code was developed using an object oriented approach, enabling the identification of individual classes or code fragments associated with a unique ID. Absence of an object oriented programmed code would affect the application of our approach since it would be challenging to identify fragments of code related to specific requirements.

If no information or only incomplete information is available about requirements and source code, we will not be able to maximize and optimize the objectives functions since that is the only input required for our approach. This restriction may affect the accuracy of the results.

In addition, the availability of more data sets that include requirements document, source code divided in classes and traceability links used for validation would strengthen the evaluation. Our use of three data sets could limit the generalizability of the study. Availability of more data
sets is necessary for performing more experimentation which would improve the generality of our findings.

4.9. Conclusions

We addressed the TRL problem using a SI algorithm. In the TRL problem, requirements and source code are the main inputs. The output is a set of traceability links from requirements to source code that maximizes the IR-based objective function defined by weighted cosine similarity. IR techniques are good techniques for finding relationships because both requirements and source code are expressed in text and both hold semantic information. We studied the generation of traceability links from requirements document to source code as a combinational problem which we solved by analyzing the search space that evaluates all possible combinations of requirements ID and source code ID (REQ, SCC) and choose the combination that maximizes the objective function.

Our research was inspired by the popularity of MS algorithms and their application in several fields. In the adapted IR-based ABC, the final set of links is generated by the iterative evaluation of the local best solutions aiming to maximize the objective function. We evaluated the results using P and R. We found values of P and R above 70% on average for data sets, EBT, Albergate and eTour.

In comparison to approaches based on genetic algorithms (Ghannem et al., 2017), the genetic algorithm NSGA-II implementation returned better results than our approach based on ABC. However, in contrast to the NSGA-II implementation which defines multiple objective functions, our approach works with minimal information from the system, which enables its application in most scenarios.
Furthermore, we found an improved balance between P and R, in comparison to RTR results using only IR techniques as found in our earlier research (Rodríguez & Carver, 2019). Future research includes expanding the study to evaluate additional data sets, application of different objectives functions and experimentation with different optimization algorithms.
5. COMPARISON OF NSGA-II AND ABC ADAPTED TO THE REQUIREMENT TRACEABILITY LINK RECOVERY PROBLEM

5.1. Introduction

As in any other engineering, software engineering (SE) has a commitment to quality and efficiency. SE defines a number of phases for high quality software development including communication, planning, modelling, construction and deployment (Pressman, 2005). These five phases assist the development of small programs, large web applications, and large complex computer-based systems. During the first phase of the development process, the communication phase, requirements of the system are gathered and analyzed. Requirements are essential in software development because they define the functionality of the systems. In order to guarantee quality, it is necessary to follow the requirements along the SE development process from the communication phase to the deployment phase. Lack of requirements tracking can cause problems in the deployed software.

Tracking requirements from the initial phases of SE to deployment is a challenging task. Often during the software development process, original requirements are modified, deleted and in common practice, new requirements are added to the system. Factors such as size of the application, number of people participating in the development process, rapid evolution of the system, and changes in original requirement documents, limit the ability of software engineers to track the requirements throughout deployment. Consequently, the final software product can be incomplete or contain errors since verifying the correct implementation of all requirements may not be possible (Ali et al., 2013; Gotel & Finkelstein, 2005; Rempel & Mäder, 2017).

Requirements traceability is a primary solution to verify that the source code of a system is consistent with its requirements (Ali et al., 2013; Gotel & Finkelstein, 2005; Rempel & Mäder,
Traceability links across textual documentation and source code resulting from the development and maintenance phases of software are helpful for several tasks, such as program comprehension, maintenance, requirements tracing, impact analysis and reuse (Pennington, 1987). However, manual generation of traceability links is a time consuming and tedious activity. In the development process, often with strict schedules, traceability links are usually not created over other activities that are considered crucial for the developers. However, empirical evidence indicates that traceability reduces the projected defect rate during software development (Rempel & Mäder, 2017; Rochimah et al., 2017).

Thus, researchers focus studies on finding novel techniques for automatic generation of links. Information Retrieval (IR) techniques has been applied to the requirements traceability recovery (RTR) problem, and they are considered as good solutions for traceability links recovery among software artifacts where textual analysis and similarity in the textual content of artifacts are fundamental for establishing relationships. However, most of IR tools and approaches are time consuming or semi-automatic (Hayes et al., 2007; De Lucia et al., 2005; Ali et al., 2011; Antoniol et al., 2002; Rodriguez & Carver 2019, Mahmoud & Niu, 2014).

Due to the popularity of the IR techniques, they have been used as baseline for many other approaches aiming to achieve automatization. Computational intelligence (CI) algorithm to assist RTR are a promising solution to achieve automatization and to find a balance between the Information Retrieval (IR) metrics precision (P) and recall (R) (Ghannem et al., 2017). Research literature presents evidence of the application of IR-based CI algorithms to the RTR (Ghannem et al., 2017; Ghannem et al., 2016).
In this research, we compare our previous IR-based computational intelligence algorithms presented in Chapters 3 and 4 of this manuscript to determine the solution that better fits the RTR problem with the goal of making the automatic RTR more efficient. We compare the Non-dominated sorting generic algorithm (NSGA-II) presented in Chapter 3, and the swarm intelligence algorithm, artificial bee colony (ABC) presented in Chapter 4. The ultimate goal is to define the algorithm that finds a better balance between experimental results of P and R, and time complexity. We address the following research questions:

RQ5-1: Which computational intelligence algorithm, NGSA-II and ABC, perform better in terms of P and R when applied to the RTR problem?

RQ5-2: Which computational intelligence algorithm, NGSA-II and ABC has the best time complexity when applied to the RTR problem?

The reminder of this research is structured as follow: section 5.2 discusses related work, section 5.3 describes the methodologies for the NGSA-II and ABC algorithms adapted to the traceability link recovery, section 5.4 explains the implementation of the algorithms, section 5.5 contains the experimental results, and section 5.6 presents conclusion and future work.

5.2. Related work

This work compares the IR-based CI algorithms presented in Chapters 3 and 4 of this dissertation to determine the approach that better fits the RTR problem based on experimental results and complexity time. The goal is to benefit the RTR problem by determining the optimal automatic solution for the generation of traceability links. We present different research efforts which aim to determine better solutions for solving the RTR problem by comparing different
techniques. We also address existing research which have investigated the RTR problem using IR-based optimization algorithms.

5.2.1. Comparing IR techniques

IR techniques have been broadly applied to the RTR problem. The research literature describes several approaches, tools and alternative solutions inspired in the application of IR techniques. To determine the best alternative, researchers compare the effectiveness of these approaches; the ultimate goal is to define the better solution to the RTR problem.

In (Mahmoud & Niu, 2014), a research that studies different IR methods applied to the requirement traceability links problem in software systems is presented. This research analyzes the performance of a number of IR methods over three traceability datasets from different domains; the goal is to define the best alternative in terms of quality of the results and browsability in comparison to basic VSM which is used as an experimental baseline. The studied methods include semantic-augmented methods such as VSM with thesaurus support (VSM-T) and Part-of-Speech-enabled VSM (VSM-POS); latent semantic methods such as LSI and latent Dirichlet allocation (LDA); and semantic relatedness methods such as explicit semantic analysis (ESA) and normalized Google distance (NGD). Experimental results show that explicit semantic methods (VSM-T, VSM-POS, ESA, and NGD) are a better alternative for recovering traceability links than latent methods (LSI and LDA). Similar to our research, this study presented by Mahmoud & Niu (2014) performs a deep analysis and compares different techniques to define the best alternative to achieve an improvement in the automatic requirements tracing problem.

In (Rodriguez & Carver, 2019), research that compares different IR techniques applied to the RTR problem is presented. This research performs the analysis based on the IR metrics of P
It also compares IR probabilistic, Vector Space Model (VSM) and Latent Semantic Index (LSI) to determine the solution that finds the highest number of correct links and the smallest number of false positives. The research presented in (Rodriguez & Carver, 2019) reaffirms the effectiveness of the application of IR techniques to the RTR problem, and it also determines the IR technique that better works for solving the RTR problem. As the research presented by Rodriguez & Carver (2019), in our research, we also compare different approaches to define the best alternative to solve the RTR.

Oliveto et al. (2010) presents another comparison approach. The research is an empirical study to statistically analyze several traceability recovery approaches based on IR techniques. Techniques are analyzed according to the Principal Component Analysis and the analysis of the overlap of the set of candidate links provided by each method. It studies the Jensen-Shannon (JS) method, VSM, and LSI, which are popular techniques commonly applied to the RTR problem. They study the Latent Dirichlet Allocation (LDA). The results suggest that JS, VSM, and LSI are almost equivalent in terms of number of correct links found, and results for LDA are lower than previously used methods. However, while JS, VSM, and LSI are almost equivalent, LDA is able to capture some information missed by the other exploited IR methods.

These research efforts seek to define which IR approach suits better the RTR problem. Additionally, they reaffirm the effectiveness of the application of IR techniques to the RTR problem which are the baseline for our optimization approach.

5.2.2. IR-base computational intelligence (CI) algorithms

A few researchers have studied the RTR problem by applying optimization algorithms. The application of computational intelligence algorithms to the traceability link recovery area has seen
some success in the automatization of the process. These algorithms use artificial intelligence to find, within a big search space, the optimal solution (list of traceability links), and the execution is controlled by the objective function and certain parameters to achieve fast convergence (Ghannem et al., 2016; Ghannem et al., 2017).

A mono-objective genetic algorithm approach which takes a set of requirements and source code of the system as input and outputs a set of traceability links between the classes of the system and the requirements is described in (Ghannem et al., 2016). A solution is based on matching a requirement, using the textual description of the requirement, with one or many classes of the source code by comparing with textual elements of the source code such as identifiers, API documentation, and comments. The approach generated the traceability links between the requirements and classes in the source code with an average P of 86% and an average R of 83%.

In (Ghannem et al., 2017), an approach is presented that uses the NSGA-II algorithm controlled by three objective functions based on semantic similarity and frequency of changes. It explodes the concept of pareto-optimal. It used the textual content of requirements document, source code and the information from the frequency of changes as the input. The results are favorable, achieving values for P and R above 60%. However, the generation of a mathematical representation based on the frequency of changes requires additional information that usually is not provided or available for the recovery of the traceability links. The researchers indicate “if no information or only incomplete information is available about the times of change of the artifacts, the approach will not be able to take advantage of the recency of change and frequency of change objectives and optimize them” (Ghannem et al., 2017). Available data sets for RTR studies often are not complete, and links need to be recovered using minimal information.
To improve results and effectiveness while using the minimal available information of the system, a multi-objective approach based on different information retrieval (IR) techniques to establish similarities is proposed in Chapter 3. This research uses only the requirements document content and the source code as the input, and the execution is controlled by the IR methods of Jaccard Similarity and a weighed cosine similarity with term frequency to find possible matches in the textual content of requirements and source code. Results are also encouraging and support the potential of applying optimization algorithms combined with IR techniques to the traceability recovery problem.

Apart from genetic algorithms, we experimented with other optimization algorithms. In Chapter 4 the application of another computational intelligence approach is proposed. We evaluated the effectiveness of the application of another population-based approach inspired by the honeybee swarm, the Artificial Bee Colony (ABC) which belongs to the swarm intelligence algorithm category. We study the RTR problem as a combinational problem. We generate different combinations of pairs formed by a requirement and source code class. The intelligent agents (bees) search for the set of pairs that maximizes the objective function. For the ABC implementation, a single objective function is defined, and it is based on the use of IR techniques to establish relationships between the textual content of requirement documents and source code. In comparison to genetic algorithms, the ABC algorithm is considered to be simpler and have faster convergence rates (Karaboga, 2005).

In this research, we evaluate and compare our previous works, the genetic algorithm, NSGA-II (Chapter 3), and the swarm intelligence algorithm, ABC (Chapter 4), to define the optimal automatic solution to the RTR problem. Both of the approaches are IR-based approaches, and they are designed to work with only minimal information of the system such as requirements
document and source code; however, they implement different optimization algorithms. The optimal implementation is the approach that finds an improve balance among P, R and time complexity.

5.3. Computational intelligence adapted to the RTR

In Chapters 3 and 4 we described our implementation of the NSGA-II and ABC algorithms to assist the RTR problem. Both algorithms were presented as an alternative to automate the RTR problem by the incorporation of computational intelligence combined with IR techniques. Even though both genetic algorithms and swarm intelligence are subcategories of the computational intelligence, they apply different internal logics for solving the same problem. Their similarities and differences enable a comparison between them.

Our previous research efforts using NSGA-II and ABC applied to the RTR problem were evaluated over three data sets, EBT, Albergate and eTour which facilitates the comparison. EBT data set consists of 40 requirements, 50 Java source code classes, and 98 trace links from requirements to classes. Albergate has 17 requirements, 55 source code classes, and 54 trace links from requirements to classes. The eTour data set includes 58 user cases use to derive the requirements, 116 source code classes, and 308 trace links from requirements to classes. Data sets were downloaded from CoESI, http://coest.org/, a website which provides data sets for traceability link recovery research. The first phase of the comparison is based on results obtained for the algorithms in term of P and R when evaluated over the three data sets.

Both algorithms were developed using Python 3.6 as the programming language. Additionally, their implementations were divided in two phases, a preprocessing phase which
prepares the input data (requirements and source code class) to be used for the algorithms, and a second phase which implements our version of the NGSA-II and ABC respectively.

The preprocessing phase, which is common for the NSGA-II and ABC, transforms the raw input data (requirements documents and source code classes) into meaningful information that becomes the input for algorithms. We manually pre-processed the data in order to keep only meaningful information. The preprocessing phase follows two paths, the requirements document path which removes stop words, punctuation, and transform capital letters to lower case letters. The source code path which extracts only the identifiers, including variables names, methods names, and class names from the source code applies after the same functions performed on the requirements phase.

Implementation of our adapted version of the NSGA-II and ABC algorithm is summarized in sections 5.4 and 5.5, respectively.

5.4. **IR-based NSGA-II algorithm**

The adapted NSGA-II version takes a set of requirements and the matching source code (classes) as input. The algorithm starts by a random generation of an initial population, controlled by the maximum size of an individual parameter, called $Max-size-I$, which represents the maximum number of pairs composing an individual $I$. Each pair is formed by exactly one requirement REQ from the requirements document and a class from the system source code SCC. $Max-size-I$ varies depending on the number of REqs and SCCs in the data set. It was set to 150 for the EBT, and Albergate and 348 for the cTour data set after experimentation on the size of the parameter.
We generated a population of possible solutions regulated by the maximum size of the population parameter, called Max-size-P, which indicates the number of individuals in the population. The population of candidate solutions that evolves towards the best sequence of RTR solutions is known as the first front. The goal is to find a sequence of individuals that maximize, as much as possible, the Semantic Similarity based on Cosine Similarity across pairs (REQ, SCC) of each individual while favoring Term Frequency across pairs of REqs and SCCs.

During each iteration $t$, we used the genetic operators of selection, crossover and mutation to generate an offspring population $Q_t$ from a parent population $P$. Next, $Q_t$ and $P_t$ were merged to create a global population $G_t$ which was evaluated using the objective functions corresponding to the two different objectives. We evaluated an individual by using textual similarity metrics such as Jaccard Similarity and a weighted cosine similarity combine with term frequency ($TF-idf$) objective functions. Equations (5.1) and (5.2) were used to calculate the value of the objective functions respectively where $m$ is the number of pairs in the individual.

$$
SemSimJS(I) = \sum_{i=1}^{m} \frac{REQ_i \cap SCC_i}{REQ_i \cup SCC_i} 
$$

(5.1)

$$
TF_{(t,d)} = \frac{\text{NumberOfTerm}(t)\text{appearintheDocument}(d)}{\text{TotalNumberOfTermsintheDocument}(d)}
$$

(5.2)

$$
idf_{(t,D)} = \log \left( \frac{\text{NumberOfDocuments}(D)}{\text{NumberOfDocumentsWithTerm}(t)} \right)
$$

$$
TFIdf_{(t,d,D)} = TF_{(t,d)} \cdot idf_{(t,D)}
$$

$$
SemSim(I) = \sum_{i=1}^{m} \text{CosineSimilarity} \left( Weight_{TF-idf}(REQ_i, SCC_i) \right)
$$
Once values for the objective functions were calculated, the solutions were sorted in order to return a list of non-dominated fronts $F$ ($F_1, F_2, ...$), where $F_1$ is the set of non-dominated solutions, and $F_2$ is the set of solutions dominated only by solutions in $F_1$. We then verified the set of non-dominated fronts starting from front $F_1$ to $F_i$ to build the next population $P_{t1}$. Since the number of solutions in all sets from front $F_1$ to $F_i$ is greater than Max-size-$P$, we used the crowded-comparison operator defined in (Deb et al., 2002) to sort the solutions of the front $F_i$, then we selected the first Max-size-$P$ solutions to guarantee we choose exactly Max-size-$P$ solutions.

The crowded-comparison operator uses non-dominated ranking which involves sorting a population $P$ based on the non-domination of individuals and the crowding distance which provides an estimate of the density of solutions surrounding that solution (Deb et al., 2002). Crowded-comparison and crowding distance are two techniques to ensure pareto-dominance and solution diversity (Deb et al., 2002). We defined termination criterion as the maximum number of generations (Max-$G$). Once the Max-$G$ parameter is met, the output returned is the set of best solutions, which is the set of solutions in the pareto-front of the last iteration. Parameters Max-size-$P$ and Max-$G$ were set to 150 and 100 respectively for EBT and Albergate, and 300 and 120 respectively for eTour.

Additionally, during the execution of the adapted algorithm, we used the genetic operators of selection, mutation and crossover. Selection in our adapted NSGA-II algorithm is based on the binary tournament selection which consists of running several tournaments by randomly selecting a pair of individuals from the population. This method provided for all the individuals equal chances of being selected, thus preserving diversity. The mutation operator randomly (based on a mutation factor) selected one or more pairs (REQ, SCC) of the individual; the pairs were replaced with other different pairs chosen randomly from the solution space. During crossover, we selected
two individuals, and we applied a double, random, cut-point crossover to create two offspring $I'_1$ and $I'_2$ from the two selected parents $I_1$ and $I_2$. More detailed information could be found in Chapter 3.

5.5. IR-based ABC algorithm

The ABC, like any other population algorithm, starts by generating the initial population which is formed by food sources. Population is regulated by the parameter Solution Number ($SN$), which is the maximum number of food source in the population. $SN$ is associated to the number of individuals evaluated at each iteration of the algorithm. It controls the number of employed bees generated since in the ABC algorithm one employed bee is related to exactly one food source. $SN$ was set to 200 for EBT, Albergate, and eTour.

We modified the standard ABC algorithm, since in our implementation the food sources are not represented by a single decimal number. Instead, they are represented by integer vectors associated with the requirements document (REQ) IDs and source code classes (SCC) IDs. The food sources are represented by a set of pairs from REQ and SCC which are randomly generated and regulated by an introduced parameter defined as the maximum size, called $Max-size-sol$. This parameter represents the maximum number of pairs composing a food source. A pair contains the number $ID$ of exactly one requirement from the requirements document and a class from the system source code. We randomly built the pairs for each food source. $Max-size-sol$ was assigned to 250 for EBT, 150 for Albergate and 400 for eTour.

The ABC algorithm uses upper and lower bounds to control the random generation of the food source. In our adaptation of the algorithm, we defined the lower bound as zero and the upper bound was related to the maximum number of requirements $ID$ and source code $ID$ found in the
data set. Integers that integrate the vector were inside the lower and upper bounds. Each vector was treated as a single solution (food source) associated to a single employed bee and having a unique value for the objective function (nectar).

In the ABC, candidate solutions evolve towards the best solutions. The goal is to find a food source that maximizes, as much as possible, the objective function. The objective function was defined by the semantic similarity based on cosine similarity across pairs (REQ, SCC). Equation (5.3) defines the formula, where $m$ is the number of pairs in the individual.

$$TF_{(t,d)} = \frac{\text{Number of times term (t) appears in the document (d)}}{\text{Total number of terms in the document (d)}}$$

$$idf_{(t,D)} = \log\left(\frac{\text{Total number of documents (D)}}{\text{Number of documents with term (t) in it}}\right)$$

$$TFidf_{(t,d,D)} = TF_{(t,d)} \cdot idf_{(t,D)}$$

$$SemSimTF(I) = \sum_{i=1}^{m} \text{CosineSimilarity}(\text{Weight}_{TF-idf}(REQ_i, SCC_i))$$

At each iteration, while we did not reach the maximum cycle number (MCN) which controls the number of iterations of the algorithms, starting with the initial population, we evaluated the food sources according to the objective function (nectar). Then, we generated new solutions for the employed bee associated to the food source using the mutation operator and reevaluated the solutions based on the objective function. We applied the greedy selection process for employed bees (employed bee phase). We calculated the probability values for the current solution so that the onlooker bee can choose the best according to its value. We assigned the onlooker bees to the solutions according to the probability, generated new solutions (mutation), evaluated the objective functions, and applied the greedy selection process for the onlooker bees (onlooker bee phase). If the solution cannot be improved after the number of times defined by the limit parameter value, the food source needed to be abandoned and the employed bee turned into a scout bee. Scout bees
in the ABC algorithm prevent inactivity of employed bee population. The Limit parameter was set to 50. At each iteration the algorithm memorizes the best solution found, and checks the termination criterion $MCN$. If $MCN$ is not satisfied, it iterates over the algorithm, otherwise the algorithm ends. $MCN$ parameters were 200 for EBT, Albergate and eTour.

During employed and onlooker phases of the ABC algorithm, a mutation operator was applied to select food sources associated to the bees to generate new solutions which were compared to the old solution to improve the objective function. If the new solution was better than the old solution, the value associated to the bee was updated to the one that holds the best value for the objective function (best nectar).

After the mutation operator, we called another function that checks that the numbers generated were within the predefined boundaries. If the resulting values were less than zero, they were set to zero, which is the lower boundary. In contrast, if the values generated were greater than the defined upper boundaries, we assigned a value chosen randomly from the solution space. More detailed information is provided in Chapter 4.

5.6. Experimental results

5.6.1. Which computational intelligence algorithm, NGSA-II and ABC, perform better in terms of $P$ and $R$ when applied to the RTR problem?

To address RQ5-1, we first evaluated our results according to the IR metrics of $P$ and $R$. High values of $P$ and $R$ are desirable. $P$ is equal to the ratio of the number of the correct links detected to the total number of links detected. $R$ is equal to the proportion of the number of links detected to the number of correct links. $P$ is related to the correctness of the results of the ABC implementation and $R$ represents the completeness of the results.
To compute P and R pair-wise, we extracted true-positive values (TP), false-positive values (FP), and false negative values (FN) for each output data set. TPs are pairs that are correctly in the output, FPs are pairs that are in the output but should not be, and FNs are pairs that should be in the output but are not. P and R defined by Eq. (5.5) and Eq. (5.6) respectively.

$$P = \frac{TP}{TP+FP} \in [0,1]$$

$$R = \frac{TP}{TP+FN} \in [0,1]$$

NSGA-II and ABC are population-based algorithms. Both start by generation of a random initial population formed by requirements (REQ) and source code (SCC) pairs (links). The random factor in the initial population causes the output of our implementation of the NSGA-II and ABC algorithms to have fluctuations at each execution. As a result, to obtain more information from the results, we iteratively executed both of our algorithms 10 times for each data set, and we computed the mean of P and R.

Table 5.1 presents the results for NSGA-II and ABC algorithms applied to the EBT data set. It shows, for each algorithm, P and R for each of the 10 iterations, the number of correct links found out the 98 traceability links in the original data set, total number of links found for each approach and the mean P and R values.

According to the mean values, the NSGA-II algorithm had better performance in term of P and R when tested over EBT. P and R were about 5% higher than the ABC for the NSGA-II algorithm. Detailed analysis indicates that, in most of the iterations, the NSGA-II algorithm outperformed the ABC algorithm in terms of number of correct links found, and values of P and R. However, for the total of links retrieved, the NSGA-II and ABC algorithms returned the same
127 values. Higher values in the NSGA-II algorithm were due to the increment of the values for number of correct links, which directly affected the values of $P$ and $R$.

### Table 5.1. $P$ and $R$ for NSGA-II and ABC for 10 iterations over EBT data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision ($P$)</th>
<th>Recall ($R$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NSGA-II</td>
<td>ABC</td>
<td>NSGA-II</td>
<td>ABC</td>
</tr>
<tr>
<td>1</td>
<td>75</td>
<td>73</td>
<td>101</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>84</td>
<td>70</td>
<td>99</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>72</td>
<td>107</td>
<td>104</td>
</tr>
<tr>
<td>4</td>
<td>67</td>
<td>75</td>
<td>80</td>
<td>103</td>
</tr>
<tr>
<td>5</td>
<td>71</td>
<td>68</td>
<td>89</td>
<td>97</td>
</tr>
<tr>
<td>6</td>
<td>68</td>
<td>70</td>
<td>96</td>
<td>99</td>
</tr>
<tr>
<td>7</td>
<td>83</td>
<td>74</td>
<td>90</td>
<td>101</td>
</tr>
<tr>
<td>8</td>
<td>74</td>
<td>69</td>
<td>105</td>
<td>99</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>71</td>
<td>102</td>
<td>98</td>
</tr>
<tr>
<td>10</td>
<td>82</td>
<td>69</td>
<td>104</td>
<td>99</td>
</tr>
<tr>
<td>Mean</td>
<td>75.5</td>
<td>71.2</td>
<td>97.3</td>
<td>97.3</td>
</tr>
</tbody>
</table>

Population-based algorithms start based on a random initial population. A better combination in the initial population has a significant impact on the output. Based on the majority of the results we found the NSGA-II algorithm as the most reliable solution for the RTR problem when evaluated with medium sized data sets such as the EBT. Figure 5.1 illustrates a comparison between NSGA-II and ABC algorithms across values of $P$ and $R$ over the EBT data.

Table 5.2 presents the results for NSGA-II and ABC algorithms performing over the Albergate data set. As in the previous data set, it shows for each algorithm, $P$ and $R$ for each of the 10 iterations, the number of correct links, the total number of links found, and the mean values for each of the mentioned indicators.
Figure 5.1. Comparison of results for the NSGA-II and ABC algorithms over EBT.

Table 5.2. P and R for NSGA-II and ABC for 10 iterations over Albergate data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NSGA-II</td>
<td>ABC</td>
<td>NSGA-II</td>
<td>ABC</td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>40</td>
<td>63</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>43</td>
<td>74</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>47</td>
<td>46</td>
<td>66</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>45</td>
<td>62</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>40</td>
<td>63</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>42</td>
<td>66</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>45</td>
<td>38</td>
<td>73</td>
<td>55</td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>42</td>
<td>70</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>49</td>
<td>43</td>
<td>71</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
<td>47</td>
<td>68</td>
<td>60</td>
</tr>
<tr>
<td>Mean</td>
<td>45.4</td>
<td>42.6</td>
<td>67.6</td>
<td>58.0</td>
</tr>
</tbody>
</table>

Based on mean values of P and R, the ABC had a better P than NSGA-II when tested over Albergate since the mean precision value was 6% higher for NSGA-II. Regarding the R values, the mean value returned for the NSGA-II algorithm was about 4% higher than the value returned
for ABC. We interpreted the results as NGSA-II returned a slightly more complete list of links associated to the R values, but ABC is more precise based on the P values when evaluated over Albergate.

In terms of number of correct links and total number of links retrieved, NSGA-II returned a higher number of correct links than ABC, and ABC returned a total number of links of 58 which was closer to the number of links in the original data set which was 54. In contrast, NSGA-II returned a total number of links of approximately 68 which increases the number of FNs. Figure 5.2 illustrates a comparison among values returned by NSGA-II and ABC algorithms of P and R over the Albergate data source code.

![Figure 5.2](image)

**Figure 5.2.** Comparison of results for the NSGA-II and ABC algorithms over Albergate.

Table 5.3 presents the results for NSGA-II and ABC algorithms performing over the eTour data set. As in the previous data set, it shows for each algorithm, P and R for each of the 10 iterations, the number of correct links, the total number of links found, and the mean values for each of the mentioned indicators.

For eTour, the mean R value was higher using ABC in comparison to NGSA-II, but the mean P value was higher for NGSA-II in comparison to ABC. A deeper analysis of the results
suggests that even though P values were higher for the NSGA-II, ABC was a better alternative for the eTour. We observed ABC returned higher values for the number of correct links and total number of links; it returned a more complete list of links falling closer to the number of links in the original data set. ACB returned approximately 35 more correct links on average than NSGA-II, and the total number of links recovered was 307 in average which was closer to the 308 links in the original data set. Figure 5.3 illustrates the graphical results for P and R over the eTour data set for the NSGA and ABC algorithm.

Table 5.3. P and R for NSGA-II and ABC for 10 iterations over eTour data set.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Correct links retrieved</th>
<th>Distinct links retrieved</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NSGA-II</td>
<td>ABC</td>
<td>NSGA-II</td>
<td>ABC</td>
</tr>
<tr>
<td>1</td>
<td>187</td>
<td>233</td>
<td>233</td>
<td>315</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>238</td>
<td>231</td>
<td>318</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>225</td>
<td>230</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>205</td>
<td>223</td>
<td>225</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>202</td>
<td>232</td>
<td>227</td>
<td>310</td>
</tr>
<tr>
<td>6</td>
<td>185</td>
<td>235</td>
<td>230</td>
<td>312</td>
</tr>
<tr>
<td>7</td>
<td>192</td>
<td>229</td>
<td>248</td>
<td>301</td>
</tr>
<tr>
<td>8</td>
<td>195</td>
<td>233</td>
<td>240</td>
<td>316</td>
</tr>
<tr>
<td>9</td>
<td>203</td>
<td>236</td>
<td>238</td>
<td>298</td>
</tr>
<tr>
<td>10</td>
<td>197</td>
<td>213</td>
<td>237</td>
<td>303</td>
</tr>
<tr>
<td>Mean</td>
<td>195</td>
<td>229.7</td>
<td>233.9</td>
<td>307.8</td>
</tr>
</tbody>
</table>
Figure 5.3. Comparison of results for the NSGA-II and ABC algorithms over eTour.

More analysis is needed to interpret experimental results, since between NSGA-II and ABC, neither algorithm performed better in all of the data sets. For EBT, NSGA-II outperformed ABC in terms of P and R. However, for Albergate and eTour, NSGA-II was more precise than ABC, but ABC had higher recall. Thus, we computed the F1 measures to define the optimal solution between the two algorithms. We assumed that values of P and R are equally important. To define the effectiveness of one approach over another approach, we used the F1 score measure which is the harmonic mean, or weighted average, of P and R scores. We used F1 as a measurement to seek a balance between P and R. It is calculated according to Eq. 5.7.

\[
F1\text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]  

(5.7)

Figure 5.4 illustrate the F1 results over EBT, Albergate and eTour. F1 values over EBT are 77.49 for NSGA-II and 71.49 for ABC. For Albergate F1 values are 74.14 for NGSA-II and 75.88 for ABC; for eTour, values are 71.63 for NGSA-II and 75 for ABC. Even though defining the optimal solution will depend on the characteristics of the data set and the parameters used to define our algorithms, based on our setup we observed that in general the ABC algorithm was a better alternative since it obtained higher F1 results for two of the three evaluated data set, Albergate and
eTour. In contrast, the NSGA-II algorithm was an optimal solution only when evaluated over the EBT. Because ABC performed better in more data sets, it is generally a better alternative for automating the RTR problem in comparison to NSGA-II.

![F1 results for the NSGA-II and ABC algorithms over EBT, Albergate, and eTour.](image)

**Figure 5.4.** F1 results for the NSGA-II and ABC algorithms over EBT, Albergate, and eTour.

According to the experimental results based on the comparison and analysis, we answer RQ5-1 by showing that ABC is the better solution for assisting the RTR problem. ABC is more stable and more probable to converge to a better solution. The graphical representation of the F1 results shows that for most data sets, results for the ABC algorithm were higher than results returned by the NGSA-II algorithm.

### 5.6.2. Which computational intelligence algorithm, NGSA-II and ABC, perform better in terms of time complexity when applied to the RTR problem?

To address RQ5-2, we analyzed our results based on time complexity. For this part of the analysis, we focus on studying time complexity of the actual implementation of the NSGA-II and ABC algorithms. We discarded any time complexity added for the preprocessing phase since the preprocessing is common to both of the algorithm.
The NSGA-II algorithm falls into the category of the Multi Objective Evolutionary Algorithms (MOEAs). For MOEAs, the upper bound time complexity is dictated by the length of the chromosomes because it can be exhaustively searched in $O(2^n)$ time, where $n$ is the number of bits required to encode the chromosome. Even though no MOEA, by definition, employs an exhaustive search of all the possible chromosome permutations, it allows a worst-case bound of $O(2^n)$ to be set for the entire class with the exception of specific algorithms proven to have a better bound (Curry & Dagli, 2014).

Faster algorithms like NSGA-II have a time complexity of $O(GMN^2)$, where $M$ is the number of objectives and $N$ the population size. However, the described time complexity refers to the computational complexity involved for a single generation of the population, not the aggregate complexity of the algorithm (Curry & Dagli, 2014). The reason the computational complexity is often stated with respect to a single generation is that it allows algorithms to be compared when the convergence rate of the overall algorithm is unknown (Curry & Dagli, 2014).

In order to calculate the actual computational complexity of an MOEA, it is necessary to know both the complexity for each generation and the number of generations. As a result, the time complexity for NSGA-II is $O(GMN^2)$, where $G$ is the number of generations (Curry & Dagli, 2014).

If we consider the time complexity time needed for solving each objective function, another factor has to be added to the overall time complexity. $F_M = O(F)$ refers to the computational complexity of each one of the objective functions. Since the NSGA-II algorithm is a multi-objective approach, $F_M$ is added according to the $M$ value and the population size at each generation. Consequently, the factor $O((GNF_M)^M)$ is added to the overall time complexity of the
NSGA-II algorithm. In our particular implementation, the M value is 2, since we have two objective functions. In that sense, the new the time complexity adapted to our implementation is \( O(GN^2 + GN^2 + GNF_1 + GNF_2) \).

Regarding the ABC algorithm, it can be divided in three main phases: the employed bee phase, the onlooker bee phase and the scout bee phase. In order to calculate the time complexity associated to the complete ABC algorithm, we first calculate the time complexity of each of the phases of the algorithm and integrate them to generate the total time complexity (Ashrafinia, 2012).

Table 5.4 illustrates the time complexity associated to the algorithm where \( F = O(F) \) refers to the computational complexity of the objective function. Additionally, \( n \) denotes the number of feasible discrete numbers between \( x_{min} \) and \( x_{max} \), and \( G, N, D \) have been already defined as the maximum number of maximum cycle number (generations), population size and the number of components in an individual vector, respectively (Ashrafinia, 2012).

<table>
<thead>
<tr>
<th>ABC</th>
<th>Employed bees phase</th>
<th>( O(Nn + NF) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onlooker bees phase</td>
<td>( O(N^2 + Nn + NF) )</td>
<td></td>
</tr>
<tr>
<td>Scout bees phase</td>
<td>( O(D + F) )</td>
<td></td>
</tr>
<tr>
<td>Single iteration</td>
<td>( O(N^2 + Nn + NF) )</td>
<td></td>
</tr>
<tr>
<td>Complete algorithm</td>
<td>( O(GN^2 + GKn + GNF) )</td>
<td></td>
</tr>
</tbody>
</table>

As Table 5.4 describes, similar to the MOEAs, for the ACB algorithm, we first calculate the time complexity associated to a single iteration of the algorithm, and then it is necessary to know both the complexity for each iteration and the maximum cycle number to define the time complexity for the entire algorithm. The maximum cycle number is referred as generation for the
NSGA-II algorithm. In order to establish a comparison, we keep the same names for the share variables such as G, N and F.

To establish a comparison, Table 5.5 presents the overall time complexity of the NSGA-II and ABC algorithm. We observe that in general, if the values for all the variables are the same, the ABC algorithm requires less time complexity to achieve the results. The main reason is that the ABC algorithm is a single objective function algorithm, and it requires fewer comparisons and evaluations than the multi objective NSGA-II algorithm. Both algorithms have a similar time complexity; however, the evaluation of the multiple objective function significantly slows the NSGA-II algorithm.

**Table 5.5.** NSGA-II and ABC algorithms time complexity

<table>
<thead>
<tr>
<th></th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA-II</td>
<td>$O(G^2N + G^2N + GNF_1 + GNF_2)$</td>
</tr>
<tr>
<td>ABC</td>
<td>$O(GN^2 + GNN + GNF)$</td>
</tr>
</tbody>
</table>

On the other hand, a detailed evaluation of the time complexity of the algorithms based on the values assigned in the experimental results show that the NSGA-II algorithm will find the optimal solution faster than the ABC algorithm. The time complexities in Table 5.5 are based on known values such as maximum size of generation or cycles and population size. These values are already defined for the algorithms, and they were chosen according to the experimentation at the lowest values that allow the algorithms to converge. Additionally, we can estimate the time complexity of the objective functions according to the length of the vector (ABC) or chromosome (NSGA-II) that needs to be evaluated.

For the EBT with the NSGA-II algorithm, the maximum size of generation and the population size were assigned to 100 and 150 respectively, while for the ABC algorithm, the values
of maximum number of cycles and population size were set to 200 and 200, respectively. Regarding the objective function evaluation, for EBT, the size of the chromosome was 150 for NSGA-II, and the size of the vector was 250 for ABC, which translates to higher time complexity associated to the objective function evaluation for ABC since it evaluates more pairs. Thus, given that the values were smaller for the NGSA-II algorithm, the NSGA-II produces a reduced time complexity in contrast to the ABC algorithm, regardless of the multi-objective evaluation.

For Albergate, the maximum generation size and the population size were assigned to 100 and 150, respectively for NSGA-II, while for the ABC algorithm, the maximum number of cycles and population size were set to 200 and 200, respectively. The size of the chromosome or vector was set to 150 for both NSGA-II and ABC. Thus, in this case since the number of the vector or chromosome is the same for both algorithms, after analyzing the time complexity formulas for NSGA-II and ABC (see Table 5.5), we discarded the time complexity associated to the value of component F2 (2nd objective function for NSGA-II) since it is equals to F (objective function for ABC) in the ABC. The new time complexities are: $O(GN^2 + GN^2 + GNF_1 + GN)$ for NSGA-II and $O(GN^2 + GNn + GN)$ for ABC. Consequently, with G and N assigned to be smaller values for the NSGA-II algorithm, its time complexity is reduced in comparison to ABC since these two parameter have the most impact on the resulting time complexity values.

For eTour, the size of the population for NSGA-II, which was assigned to 300, is larger than the population size for ABC which was 200. On the other hand, the number of generations of 120 for NSGA-II is smaller than the maximum number of cycles for ABC, assigned to 200. In addition, for eTour the size of the chromosome was 348 for the NGSA-II, and the vector size was 400 for the ABC. According to this setup, we estimate that both algorithms have a similar time complexity. However, a more detailed analysis of the parameter and time complexity shown in
Table 5.5 shows that, for eTour, the ABC would be faster since the value assigned to the population size (N) is significantly smaller than the value assigned to the equivalent parameter for NSGA-II. The time complexity formula for NSGA-II shows that the parameter N has the highest impact on the time complexity value. N was assigned to be a larger number for NSGA-II than ABC in the setup of the eTour data set. Also, given that ABC evaluates a single objective function instead of two as NSGA-II, will affect positively the time complexity regardless the higher values assigned to number of cycles and the size of the vector.

Since NSGA-II use smaller values for the parameters in two of three data sets in comparison to ABC, the NSGA-II required lower time complexity to achieve results. To answer RQ5-2, we found that even though the NGSA-II algorithm evaluates two objective functions, it returned the optimal solution faster, since values assigned to the defined parameters (N, G, and F) for NSGA-II, in the majority of the cases, are significant smaller than ABC, decreasing the impact of the multi-objective evaluation.

Time complexity of the NSGA-II and ABC, presented in Table 5.5, shows that all terms are affected by parameters G and N, which have a larger impact on the value of the time complexity for ABC since they were assigned to be larger numbers in the ABC setup for the majority of the cases. Table 5.6 presents the execution time obtained for our approaches over each of the data sets. Results reaffirm our interpretation of the time complexity formulas for NSGA-II and ABC.

<table>
<thead>
<tr>
<th></th>
<th>EBT</th>
<th>Albergate</th>
<th>eTour</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA-II</td>
<td>2026.69025627</td>
<td>1319.7475</td>
<td>20280.18368794</td>
</tr>
<tr>
<td>ABC</td>
<td>4366.50462290</td>
<td>1611.16283884</td>
<td>10299.06149262</td>
</tr>
</tbody>
</table>
Based on the evaluation of the time complexity of the NSGA-II and ABC algorithms, we found that in general the ABC algorithm has a more efficient time complexity when compared to the NSGA-II algorithm. However, in the particular case of our implementation, the NGSA-II algorithm is the faster option because the numbers associated to the parameters involved in the time complexity calculation were smaller than the values required for the ABC algorithm in the majority of our data sets. Consequently, the experimental results show that the ABC algorithm had better P and R values, but it was slower when compared to the NSGA-II algorithm.

5.7. Conclusion

The popularity and efficiency of metaheuristic algorithms was an inspiration for this research. Search-based optimization algorithms iteratively evaluate sets of information to find the set that maximizes the objective functions. In our study for solving the RTR problem, requirements and source code are the main inputs. The output is a set of traceability links from requirements to source code that maximizes the IR-based objective functions defined by textual similarity. The final set of links is generated by the random and iterative generation of sets of lists of requirements and source code (REQ, SCC) evaluated based on the objective functions. Many approaches, which are based on optimization algorithms, have been developed to enhance the RTR problem. The literature review shows the application of genetic and swarm intelligence algorithms to the RTR problem. To identify the better algorithm for achieving automatization while maintaining quality and efficiency in the RTR problem, we compared two IR-based optimization algorithms to find the one that better addresses the RTR problem.

We studied a genetic algorithm (NGSA-II) and a swarm intelligence algorithm (ABC), both controlled by IR objective functions. The criteria to establish the comparison were P, R, and
time complexity. Each research question addressed one of the criterion of the comparison. Experimental results showed the ABC algorithm outperformed the implementation of the NGSA-II for P and R values. The results were more precise and complete for ABC than for NGSA-II. However, analysis of time complexity showed that, based on the defined parameters NGSA-II had a better time complexity than ABC since values for the parameter were smaller for NSGA-II. More experimentation on the parameter could improve the time complexity of the ABC algorithm.

When prioritizing values of P and R over time complexity, ABC is a better option when applied to the RTR problem. It is a good alternative to automatize the process while offering better quality results. Additionally, it works with minimal information of the system.
6. CONCLUSIONS AND PERSPECTIVES

6.1. Conclusions

We have presented two methods for improving RTR. The study first analyzes the application of IR techniques to the RTR problem. After applying the most popular IR techniques to the RTR problem, we found that the application of IR techniques is beneficial for the RTR problem since the textual content of the system artifacts holds abundant information about the system. We use the textual content to establish matches based on textual similarity among requirements and source code classes, which is the main focus of this research. The high P values obtained by the application of IR techniques to RTR reinforce the advantages of these techniques to find links among systems components expressed in text, but the R is low since the completeness of the results is not guaranteed. Our results indicate that IR techniques are a good alternative for solving the RTR problem, but they need to be combined with more intelligent approaches to facilitate the generation of more complete list of links.

The analysis of IR techniques and the success of their application to the RTR problem in terms of P is followed by the presentation of an innovative approach that studies the RTR problem as an optimization problem where the objective functions that need to be maximized are based on IR techniques. We applied the genetic algorithm NGSA-II combined with IR techniques to guide the search for all combinations of links formed by requirements (REQ) and classes (SCC) with the higher textual similarity. Results are promising for the study of the RTR problem as an optimization algorithm. We assumed that links with higher similarity are more likely to be related and that the NSGA-II algorithm evolves to that optimal set of links. Our approach returned values above 70% for both P and R.
In addition to the IR techniques, we investigated another innovative optimization approach, the application of Artificial Bee Colony (ABC) to the RTR. The ABC, which is also controlled by IR techniques, is widely applied to combinational problems. In this research, we took advantage of the combinational factor in the ABC to perform a deep search aiming to find more positive matches. Our results show that ABC does a more exhaustive search than the NSGA-II, and, as a result, improves the values of P and R.

Our results obtained by the application of NSGA-II and ABC algorithms to the RTR problem, which achieve P and R values above 70%, result in a positive answer to RQ1, RQ2 and RQ4. They show that the application of an intelligent search approach is a good alternative for the RTR problem in terms of better results and automatic generation of links. Additionally, the NSGA-II and ABC results show that the combination of IR techniques and optimization algorithms is a promising area of study. Multi-objective genetic and swarm intelligence algorithms enable an exhaustive search, which explores the search space to find pairs of REQ and SCC with higher textual similarity values evaluated according the IR techniques. Thus, optimization algorithms guide the search space, and IR techniques control the search process.

Our results for the IR-based NSGA-II and ABC implementations demonstrate that the application of IR techniques combined with the optimization algorithm is more effective in contrast to the basic application of IR techniques. Our results provide a positive answer for RQ3 and RQ5. When compared with a basic implementation of the IR techniques, our adapted NSGA-II and ABC approaches obtained better results in terms of recall since they allow the generation of more complete lists of links, while maintaining good values of precision. We calculated the F1 measure, assuming that values of P and R are equally important. Our intelligent approaches outperformed our basic implementation of IR techniques by approximately 30% on average.
Both NSGA-II and ABC approaches were validated over the same data sets EBT, eTour and Albergate, facilitating a comparison between them. Comparisons are necessary to determine the best solution between the two approaches when solving the same problem. As part of our research, to answer RQ6, we compared the effectiveness of the two IR-based optimization algorithms to define the best alternative to be applied to the RTR problem. Our comparison is based on values of P and R, but also considers the time complexity of both algorithms to identify the fastest option. We found that NGSA-II is faster and ABC is more precise. Assuming that values of P and R are more important than time complexity for the RTR problem, we find that ABC is the best alternative.

6.2. Contributions

The major contributions from this research are the development of efficient new methodologies for automatic traceability link recovery which represent improvements for the RTR problem. We studied a field that has not yet been extensively explored in the literature which is the application of optimization techniques, such as metaheuristic natural-inspired algorithm, to the RTR problem. In particular, the contributions are as follows:

- We introduced an intelligent approach for the RTR problem based on the application of an IR-based NSGA-II algorithm to enable the automatic recovery of traceability links between the requirements document and the source code of the system by matching the textual content of both artifacts and finding the links with higher textual similarity. Our approach relies on using only minimal information of the system (textual content of requirements and the source code), whereas other approaches require more input information. Results
showed that we retrieved traceability links across requirements and source code classes with values of P and R above 70%.

- We introduced the application of another optimization approach, a swarm intelligence algorithm, to the RTR problem to allow the recovery of traceability links between requirements and source code of the system by establishing possible links based on the textual similarity of both artifacts. Results indicated that the application of ABC is beneficial for the RTR problem since it retrieved links between requirements, and source code classes with P and R values above 70%. This is a novel approach because we found no evidence of its application for the RTR problem.

- We analyzed the application of basic IR techniques in comparison to the application of IR-based intelligent approaches. The results showed that the intelligent approaches (optimization algorithms) outperformed the results of the basic implementation of IR approaches.

- We analyzed the application of our IR-based intelligent approaches in comparison to existing intelligent approaches found in the literature. Results showed that even though some solutions achieved higher values of precision and recall, our IR-based approaches were able to find traceability links between requirements and source code using only their textual content while maintaining high values of precision and recall. We presented a solution that could be used when only minimal information (requirements and source code) of the system is available.

- We conducted a comparative study between the IR-based NSGA-II and IR-based ABC to identify the algorithm that better suits the RTR problem. Results show that NSGA-II is
more time efficient, but the ABC implementation is better in terms of P and R since it finds a more precise and complete list of traceability links than the NSGA-II.

Overall, the contribution of this research is to demonstrate based on experimental results that IR-based optimization algorithms (intelligent approaches) are beneficial for the RTR problem. Our results show that the application of optimization algorithms to the RTR problem generates a more complete list of requirement traceability links with higher values of prevision and recall when compared to IR techniques.

6.3. **Future perspective**

Results are dependent on the parameters; in the future, we plan to study different parametrization in both algorithms since it could tune the results. Additionally, we plan to expand the experimentation with more data sets to generate a more standard algorithm that could be adapted to more general cases.

Moreover, we plan to expand our research to evaluate more optimization algorithms as well as different objective functions; it will be informative to define the objective function based on more advanced techniques capable of identifying synonyms or predictions using machine learning and deep learning. A contribution of this research is the ability to recover links using minimal information about the system. However, we plan to use additional information from the system to create traceability links to improve P and R. The eventual goal is to achieve 100% accuracy and completeness. Thus, a better representation of the problem using more information could increase the results.
Another promising approach would be the recovery of links among other artifacts of the system, and the creation of an application that allows the user to select which artifacts to use for the recovery of the links.
APPENDIX A. COPYRIGHT INFORMATION

3/23/2020

Comparison of Information Retrieval Techniques for Traceability Link Recovery
Author: Danissa V. Rodriguez
Publisher: IEEE
Date: March 2019
Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © [year of original publication] IEEE.
2) In case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author’s approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication].
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
3) In placing the thesis on the author’s university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of (university/educational entity’s name here)’s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE

© 2020 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Terms and Conditions
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

https://180.copyright.com/AppDispatchServlet?form=Top

1/1

148
APPENDIX B. INSTITUTIONAL REVIEW BOARD APPROVAL

ACTION ON EXEMPTION APPROVAL REQUEST

TO: Danissa Rodriguez Caraballo
Computer Science

FROM: Dennis Landin
Chair, Institutional Review Board

DATE: February 6, 2020

RE: IRB# E12021

TITLE: Multi-objective Optimization for Requirement Traceability Recovery


Review Date: 2/5/2020

Approved X Disapproved

Approval Date: 2/5/2020 Approval Expiration Date: 2/4/2023

Exemption Category/Paragraph: 4a

Signed Consent Waived?: N/A

Re-review frequency: Three years

LSU Proposal Number (if applicable):

By: Dennis Landin, Chairman

PRINCIPAL INVESTIGATOR: PLEASE READ THE FOLLOWING – Continuing approval is CONDITIONAL on:

1. Adherence to the approved protocol, familiarity with, and adherence to the ethical standards of the Belmont Report, and LSU's Assurance of Compliance with DHHS regulations for the protection of human subjects*
2. Prior approval of a change in protocol, including revision of the consent documents or an increase in the number of subjects over that approved.
3. Obtaining renewed approval (or submittal of a termination report), prior to the approval expiration date, upon request by the IRB office (irrespective of when the project actually begins); notification of project termination.
4. Retention of documentation of informed consent and study records for at least 3 years after the study ends.
5. Continuing attention to the physical and psychological well-being and informed consent of the individual participants, including notification of new information that might affect consent.
6. A prompt report to the IRB of any adverse event affecting a participant potentially arising from the study.

SPECIAL NOTE: When emailing more than one recipient, make sure you use bcc. Approvals will automatically be closed by the IRB on the expiration date unless the PI requests a continuation.

* All investigators and support staff have access to copies of the Belmont Report, LSU's Assurance with DHHS, DHHS (45 CFR 46) and FDA regulations governing use of human subjects, and other relevant documents in print in this office or on our World Wide Web site at http://www.lsu.edu/irb
REFERENCES


150


Faiz, F., Easmin, R., & Gias, A. U. (2016, September). Achieving better requirements to code traceability: Which refactoring should be done first?. In 2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC) (pp. 9-14). IEEE.


VITA

Danissa V. Rodriguez Caraballo was born in Margarita Island (Venezuela). She received her Bachelor degree in Computer Science from Universidad de Oriente in 2010 with a special mention of Magna Cum Laude, graduating as the student with the highest GPA from the class of 2010 formed by more than 2000 students. During the graduation ceremony she was elected by the university authorities to give the speech in the 2010 fall graduation as a recognition of her excellence. During her undergraduate days she was recognized with multiple scholarships for academic excellence, and she represented the university in different events.

She initiated her professional career working as an Assistant Professor for the Universidad de Oriente. She taught in the Computer Science Department for four years, and her duties included teaching different classes from the curriculum, working on undergraduate research projects, and developing new and innovative subjects to integrate the curriculum of the undergraduate program of computer science. As a professor, she discovered one of her passion, and found the teaching experience very rewarding. She was one of the youngest professors in the department, and she was rewarded with a promotion from the university after three years of a great performance.

In 2014, she received a Fulbright scholarship to continue her education in the United States becoming the first person who received this award in her home town. After receiving a Fulbright scholarship, she started graduate studies in the Computer Science Department at Louisiana State University where she joined the Software Engineering Laboratory of Professor Doris Carver in May 2015. She received a Master of Science degree in Computer Science from Louisiana State University in August 2016, and she expects to receive the Doctor of Philosophy degree in Computer Science in August 2020. To date, her work has been presented in two major international conferences. Her motto is “If you can imagine it, you can make it happen”.