Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Specialized Path-based Technique to Test IoT System Functionality under Limited Network Connectivity

Doctoral Thesis

Ing. Matěj Klíma

Prague, April 2023

Ph.D. programme: Informatics
Branch of study: Computer Science

Supervisor: doc. Ing. Miroslav Bureš, Ph.D.
Supervisor-Specialist: Dr. Bestoun S. Ahmed Al-Beywaynee, Ph.D.

Thesis Supervisor:
doc. Ing. Miroslav Bureš, Ph.D.
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13
121 35 Prague 2
Czech Republic

Thesis Supervisor-Specialist:
Dr. Bestoun S. Ahmed Al-Beywaynee, Ph.D.[1]
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13
121 35 Prague 2
Czech Republic

[1]Also with Dept. of Mathematics and Computer Science, Faculty of Health, Science and Technology, Karlstad University, Sweden

Acknowledgements

I am deeply grateful to my thesis supervisor doc. Ing. Miroslav Bureš, Ph.D., who influenced my studies immensely. He introduced me to the system testing domain years ago when I was working on my bachelor's thesis. During my doctoral studies, he provided me with the necessary background information on the current state of research and the general rules of research work.

Secondly, I would like to offer my special thanks to Prof. Angelo Gargantini of the University of Bergamo, Italy, who hosted me for some time at his research institution and provided me with much advice for my research.

I would like to thank also Dr. Bestoun S. Ahmed Al-Beywaynee, Ph.D., who provided important suggestions and improvements to my research work.

I gratefully acknowledge the support of funding sources that made this Doctoral Thesis possible. The research was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/177/OHK3/3T/13 *Algorithms and solutions for automated generation of test scenarios for software and IoT systems.*

Finally, I would like to express my gratitude to my parents and my partner Vendulka. Without their understanding and encouragement over the past few years, it would be impossible for me to complete my study.

Abstract

Existing Internet of Things (IoT) systems encounter several reliability-related issues, among which the dynamic behavior of IoT systems under limited and unstable network connectivity is yet to attract significant research attention. Several ad hoc approaches can be employed to intuitively test this behavior. However, the effectiveness of ad hoc methods for defect detection and the overall expenditure for testing limited network connectivity remain unclarified. Therefore, this thesis presents a new specialized path-based technique to test the processes of an IoT system in scenarios with limited or disrupted network connectivity affecting these processes.

This technique can be scaled using four levels of test coverage criteria to determine the strength of the created test scenarios. Additionally, we proposed three algorithms to generate test cases for implementing the technique: breadth-first search graph traversal-based test case composition, ant colony optimization-based search, and genetic-algorithm-based test case composition. Subsequently, the effectiveness of the proposed approach was compared with that of the two baselines. The first baseline comprised three standard path-based testing approaches—Edge, Edge-pair, and Test Depth Level (TDL) 3 coverage—applied to the discussed case. The second baseline was a possible solution utilizing a standard path-based testing approach based on the test requirements, in which the test scenarios are computed by a set-covering algorithm.

In the experiments, the proposed algorithms were implemented in an Oxygen platform, which is an experimental model-based testing environment. To compare the effectiveness of the algorithms, we defined 310 problem models comprising real-life project models, artificially created models resembling the topology of real systems, and purely artificially generated models offering diverse problem instances. Furthermore, we introduced artificial defects into the models to evaluate the potential effectiveness of the individual algorithms in detecting limited network connectivity related defects present in the system.

Although the ant colony optimization-based method yielded the best results for most problem instances, the other two algorithms provided the best results for certain portions of the problem instances. In certain cases, even the baseline test requirements-based approach delivered the best results, an effect relatively common in the path-based testing field in general. Therefore, all the compared algorithms were combined into a portfolio strategy to ensure the generation of the best test set.

Overall, the proposed technique is applicable to numerous cases in which an IoT system is operating under limited network connectivity, especially for testing mission-critical IoT systems. Furthermore, the technique can be generalized to test scenarios in which a system component undergoes failure, disconnection, or damage.

Keywords: Internet of Things, Reliability, Path-based Testing, Model-based Testing, Test Automation, Limited Network Connectivity, Test Case Generation.

Anotace

Současné projekty systémů internetu věcí (IoT) se často potýkají s problémy souvisejícími s bezpečností a spolehlivostí, mezi kterými hraje výraznou roli chování systému při omezeném či nestabilním síťovém připojení. Přes jeho význam tomuto problému doposud nebyla v oblasti výzkumu věnována dostatečná pozornost. Pro testování tohoto chování lze použít různé ad hoc přístupy, avšak je obtížné odhadnout jak jejich účinnost, tak jejich reálnou nákladnost. Tato práce proto představuje novou techniku založenou na průchodech procesů systému (path-based testing) přímo zaměřenou na testování scénářů s omezeným či narušeným síťovým připojením ovlivňujícím tyto procesy.

Tato technika může být škálována pomocí čtyř úrovní kritérií pokrytí testů, které určují účinnost a cenu vytvořených testovacích scénářů. Dále jsme navrhli tři algoritmy pro generování testovacích scénářů z modelu testovaného systému: algoritmus založený na prohledávání grafu do šířky, algoritmus založený na simulovaném chování mravenčí kolonie a adaptace genetického algoritmu. Následně jsme porovnali účinnost navrženého přístupu s účinností dvou existujících způsobů generování testovacích scénářů. První způsob zahrnoval tři standardní v průmyslu používaná kritéria pokrytí testů: pokrytí hran, pokrytí dvojic hran a kritérium Test Depth Level 3. Druhým srovnávaným způsobem bylo využití předchozího algoritmu založeného na tzv. testovacích požadavcích.

Pro experimentální ověření byly navržené algoritmy implementovány v rámci platformy Oxygen. Abychom porovnali efektivitu algoritmů, vytvořili jsme 310 různých modelů testovaného systému. Část těchto modelů pocházela z reálných projektů vývoje IoT systémů, část byla vytvořena uměle tak, aby modely svojí topologií připomínaly reálné systémy a poslední část modelů byla pro větší různorodost vygenerována čistě uměle. Dále jsme do modelů zavedli umělé defekty způsobené omezeným síťovým připojením, abychom vyhodnotili potenciální efektivitu jednotlivých algoritmů při detekci těchto defektů.

Pro většinu modelů systému v těchto experimentech přinesla nejlepší výsledky metoda založená na simulaci chování mravenčích kolonií. Pro určitou menší část modelů však poskytly nejlepší výsledky i zbylé dva navržené algoritmy. V malém počtu případů dosáhl nejlepšího výsledku dokonce i existující přístup založený na testovacích požadavcích. Výsledek není překvapivý, jedná se o relativně běžnou situaci v oblasti testování procesů systému. Abychom tedy zajistili generování nejlepší sady testovacích scénářů pro co nejširší spektrum různých modelů systému, zkombinovali jsme všechny porovnávané algoritmy do tzv. portfoliové strategie.

Navržená technika je dobře použitelná v mnoha situacích, kdy IoT systém funguje s omezenou síťovou konektivitou, zejména pro testování kriticky důležitých systémů IoT. Techniku lze dále zobecnit na testovací scénáře, ve kterých dojde k selhání, odpojení nebo poškození konkrétní součásti systému.

Klíčová slova: Internet věcí, spolehlivost, testování na základě cest, testování na základě modelu, automatizace testování, omezené síťové připojení, generování testovacích případů.

List of Tables

4.1 Test case evaluation criteria $\mathcal{E}$. 30
4.2 The values of the constants for the ANT algorithm. 36
4.3 The initial values of the variables and the values of the constants for the AGA algorithm. 49
4.4 Time and space complexity of the proposed algorithms. 59

7.1 Existing third-party IoT systems, whose models were used in experiments. 72
7.2 Overall properties of SUT models used in experiments. 76
7.3 Complete overview of SUT models used in experiments (Part 1). 77
7.4 Complete overview of SUT models used in experiments (Part 2). 78

8.1 Average values of test set evaluation criteria over all G for the *AllBorderCombinations* and *EachBorderOnce* test coverage criteria. 81
8.2 Average values of test set evaluation criteria over all G for the *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* test coverage criteria. 85

List of Figures

2.1 Example of an SUT problem model, a set of test requirements, and a resulted set of test paths. 7

4.1 Initial example of an SUT affected by a network connectivity outage in its submodule and a path-based test case enabling the test of the SUT behavior in such situations. 23
4.2 Illustration of defined model elements in exemplary IoT system. 25
4.3 High-level view at the DTA functionality in a defense operation. 27
4.4 UML diagram of DTA with stealth mode process. 27
4.5 An LNCT example problem model of a DTA with stealth mode process. . 28

7.1 Sample SUT model with highlighted LCZs in the Oxygen application. . . . 70
7.2 Highlighting selected test cases in a SUT model in the Oxygen application. 71
7.3 Distribution of sources of SUT models used in experiments. 74
7.4 Visualization and manual definition of artificial defects in Oxygen application. 75

8.1 Algorithm comparison for *AllBorderCombinations* test coverage criterion through the test set evaluation criteria. 82
8.2 Algorithm comparison for *EachBorderOnce* test coverage criterion through the test set evaluation criteria. 83
8.3 Algorithm comparison for *ComprehensiveAllBorderCombinations* test coverage criterion through the test set evaluation criteria. 86
8.4 Algorithm comparison for *ComprehensiveEachBorderOnce* test coverage criterion through the test set evaluation criteria. 87
8.5 The visualization of the number of SUT models for which the individual algorithms produced the best T (part 1). 91
8.6 The visualization of the number of SUT models for which the individual algorithms produced the best T (part 2). 92

List of Acronyms

ABC Artificial Bee Colony. 14

ACO Ant Colony Optimization. 11, 12, 14, 15, 35, 109

AGA Adapted Genetic Algorithm-based paths generation. 31, 44–49, 51, 59, 63, 67, 69, 71, 76, 79–81, 84, 85, 88–90, 93–99, 105, 109–111

ANT Ant Colony Optimization-based algorithm. 31, 35–37, 39, 63, 67, 69, 71, 76, 79, 80, 84, 88–90, 93–101, 105, 109–111

CFG Control Flow Graph. 9, 11–14

COP Edge connection outage probability. 24, 72

CRUD Create, Read, Update, Delete. 8

DFT Data-flow Testing. 8, 9, 11, 13, 47, 112

DTA Digital Triage Assistant. 26

GA Genetic Algorithm. 12–15, 46–49

LCZ Limited Connectivity Zone. 18, 24, 25, 29–32, 34–47, 49–51, 53, 55–60, 64, 65, 70–75, 93, 95, 100, 101, 110

LNCT Limited Network Connectivity Test. 2, 3, 21, 31, 67, 70, 73, 103, 104

LTE Long Term Evolution. 16

MBT Model-based Testing. 5, 11, 17, 69, 103, 107

OSI Open Systems Interconnection. 16

PSO Partical Swarm Optimization. 14

SBSE Search-based Software Engineering. 10

SBST Search-based Software Testing. 10

SPC Shortest Paths Composition. 31, 63, 67, 69–71, 76, 79, 80, 84, 88–90, 93–100, 105, 109–111

SUT System Under Test. xvi, 2, 3, 5, 6, 8–17, 20–26, 29–31, 33, 34, 37–41, 43, 46, 49, 50, 52, 53, 55, 58, 60, 64–71, 73–76, 79, 89, 90, 93–101, 103–106, 109–111

TDL Test Depth Level. 2, 63, 64, 80, 81, 85, 89, 94, 96, 98, 102, 106, 111

TR Test Requirements-based algorithm. 63–65, 67, 69, 76, 79, 80, 84, 89, 90, 93–97, 99, 105, 109–111

UML Unified Modeling Language. 12, 13, 15, 17, 69

Contents

List of Tables **ix**

List of Figures **xi**

List of Acronyms **xiii**

1 Introduction **1**

2 Related Work **5**
2.1 Model-based Testing and Used Models 5
2.2 Preliminaries and the Path-based Testing 6
2.3 Data-flow Testing 8
2.4 Analysis of Existing Algorithms and Strategies 9
2.4.1 Search-based Software Engineering Strategies 10
2.4.2 The Ant Colony Optimization Approach 11
2.4.3 The Genetic Algorithm Approach 12
2.4.4 Other Nature-inspired Approaches 14
2.5 Alternative Techniques for Testing IoT System Functionality with a Limited Network Connectivity 15
2.6 Summary of Related Work and Motivation 17

3 Thesis Statement and Research Questions **19**

4 Proposed Limited Network Connectivity Technique **21**
4.1 Model of the Problem 24
4.1.1 Problem Model Transformation Guidelines 25
4.1.2 Problem Model Transformation Example 26
4.2 Test Coverage Criteria 29
4.3 Test Set Evaluation Criteria 30
4.4 Proposed Algorithms 31
4.4.1 Shortest Paths Composition Algorithm 31
4.4.2 Ant Colony Optimization-based Algorithm 35
4.4.3 Adapted Genetic Algorithm 44
4.4.4 Complexity of the Proposed Algorithms 59

5 Baseline Algorithms **63**
5.1 Initial Baseline 63
5.2 Test Requirements-based Algorithm 64
5.2.1 The Main Algorithm 64

5.2.2 Extraction of Test Requirements 65

6 Portfolio Strategy **67**

7 Methods of the Experiments **69**
7.1 Implementation of Algorithms 69
7.2 Sources of System Under Test (SUT) Models used in Experiments 71
7.3 Simulation of System Defects 73
7.4 Detailed SUT Models Properties 75
7.5 Computation of Test Cases 76

8 Results of the Experiments **79**
8.1 Properties of Test Sets Produced by Compared Algorithms 79
8.2 Algorithms that Produced the Best Test Sets for Particular SUT Models . 89
8.3 Effectiveness of Limited Network Connectivity Defects Detection in the SUT 93
8.4 Results of the Portfolio Strategy 94
8.5 Time Effectiveness of the Algorithms 96

9 Discussion **99**

10 Practical Applicability of the Proposed Technique **103**

11 Threats to Validity **105**
11.1 Internal Validity Threats 105
11.2 External Validity Threats 106

12 Conclusions **109**
12.1 Future Directions 111

A List of Publications Related to Thesis Topic **113**
A.1 Published IF Journal Papers 113
A.2 Published Conference Papers 113
A.3 Registered Patents 114
A.4 IF Journal Papers Under Review 114

B List of Other Publications Related to IoT Testing **115**
B.1 Published IF Journal Papers 115
B.2 Published Conference Papers 115
B.3 Registered Utility Models 116
B.4 Patent Applications Under Review 116

Bibliography **126**

Chapter 1

Introduction

In the past decade, IoT systems have significantly advanced from an initial hype to a daily-life technological reality that impacts individuals' work processes and lifestyle [1]–[4]. This growth has created several challenges in terms of quality, usability, security, and reliability of these systems, especially for those with a mission-critical nature [5]–[8]. Among these issues, the reliable functionality of a dynamic IoT system becomes critical if the system components or its processes are operating with limited or unstable network connectivity [5], [6]. In this thesis, limited connectivity may refer to a complete network connectivity outage, intermittent connectivity, significantly low bandwidth, high network error rate, or any state that might negatively influence the reliability and functionality of an IoT system.

Why are the IoT systems specific at this point? In case of the commonly used web-based client-server architecture of software information systems, the server side (back-end) is generally static in the spatial sense and connected to a stable network. In contrast, the client side may physically shift and be subject to limited network connectivity or even temporary network outages. This situation is typical in rural or sea areas with weak or no wireless network coverage or in tunnels in urban areas. In such scenarios, users are prepared to tolerate network connectivity issues and interact with the system accordingly. However, in case of dynamic IoT systems, connected devices such as sensors, actuators, and even the back-end infrastructure can be spatially displaced, and they are more sensitive to limited or disrupted network connectivity. Examples of dynamic sensor networks in which the geographical location of devices varies during system operation include systems in smart farming, smart cars, intelligent transportation, and defense and logistics systems [9], [10].

As such, the reliability of the service provided by the system to its users must be maintained and the system behavior should remain deterministic even if the IoT system or its components experience limited or disrupted network connectivity. Although users may

accept the restricted system functionality in such scenarios, they must be timely notified and the system cannot interrupt actual transactions, lose data, enter an unexpected state, crash, or become unresponsive. In dynamic IoT systems subjected to network connectivity limitations, their functionality must be tested under these conditions. Accordingly, optimal testing must be performed to ensure effective detection of relevant defects and reduce the costs associated with such testing. This economic aspect is one of the primary motivations of the following proposal. In addition, the desired reliability and safety of mission-critical IoT systems under limited network connectivity may be achievable only with a proper test automation method.

This thesis proposes the Limited Network Connectivity Test (LNCT)—a novel and specialized technique that generates test scenarios to test the functionality of an IoT system under limited network connectivity. In principle, the LNCT is based on the established path-based testing discipline [11], [12] and defines a specialized SUT model for implementing the technique. Based on this model, we proposed three novel algorithms for automated test-case generation and four baselines to compare the performance of the proposed algorithms.

The baselines included an Edge, Edge-pair, and Test Depth Level (TDL) 3 established test coverage criteria, and a solution using a standard path-based testing approach, based on the established test requirements concept and test paths computed by a set-covering algorithm proposed by Li *et al.* [13].

The effectiveness of all three proposed algorithms was evaluated based on the properties of the generated test cases with several criteria considering the number of test cases, their length, ratio of unique test steps, and others (detailed in Section 4.3). Moreover, we evaluated the defect detection potential of the generated test cases and compared their efficiencies with the baselines.

The contributions of this dissertation are summarized as follows:

1. Formulation of a problem model and four novel test coverage criteria addressing the testing problems for IoT systems operating with limited network connectivity.

2. Proposal of three algorithms to generate test scenarios for this testing problem.

3. Formulation of four baselines and a comparative analysis of their performance with that of the proposed algorithms.

4. Conducting an evaluation study with 310 experimental SUT models, wherein the properties of the generated test scenarios were compared in terms of the testing expenditure, based on the number of test steps and the potential of test cases for detecting the defects in a SUT.

5. Introduction of a portfolio strategy that combines all proposed algorithms to yield the best result for various possible SUT models that might occur in industrial practice.

The remainder of this thesis is organized as follows: the related literature is analyzed in Section 2, and the motivation of this research is summarized in Section 2.6. Based on this motivation, the thesis statement and research questions are elaborated in Section 3. Thereafter, the principles of the proposed technique are described in Section 4. In addition, the SUT model, test coverage criteria, and test set evaluation criteria are introduced in Sections 4.1, 4.2, and 4.3, respectively. Thereafter, the three novel algorithms proposed for generating test sets from the SUT model are discussed in Section 4.4, and the baseline algorithms used in the present experiments are described in Section 5. Subsequently, in Section 6, the three proposed algorithms were combined with one of the baselines to formulate a portfolio strategy for obtaining the best test set for a given SUT model.

Furthermore, the experimental design, including the implementation of the proposed algorithms, sources and properties of experimental SUT models, and the simulation of limited network connectivity-related defects of SUT instances are presented in Section 7. Thereafter, the experimental results are detailed in Section 8. In particular, the results of the individual algorithms were analyzed, followed by the analysis of the portfolio strategy and algorithm run times. In Section 9, we discussed the results, analyzed the vital findings, and inferred conclusions from the data. Moreover, the practical applicability of the LNCT is highlighted in Section 10, wherein the proposed LNCT was extended to a broader range of testing tasks. The threats against validity and taken steps of minimizing their possible impact are analyzed in Section 11. Finally, the major findings of this thesis and the new research stream opened by this Ph.D. project are summarized in Section 12.

Several parts of this thesis were published in our previous papers, namely an initial study describing the concept of LNCT [14], and a study comparing two of the presented algorithms with a baseline [15]. One of our real IoT projects used as the source of system models in the experiments was also described in a dedicated article [16].

Another study, comparing a genetic-algorithm-based solution with baselines, titled *Genetic Algorithm for Path-based Testing of Component Outage Situations in IoT System Processes*, is now under review in IEEE Internet of Things journal.

The concept of LNCT and an initial algorithm computing the test cases were registered as United States patent 1,194,700 B2 and is also registered as United Kingdom patent GB2594346.

Some parts of the text in this thesis are based on the descriptions in these publications or documents and may partially overlap.

Chapter 2

Related Work

Relevant to the current research topic and proposal, eight directions of related literature must be analyzed: (1) general Model-based Testing (MBT) context, (2) path-based testing preliminaries and related techniques and algorithms, and (3) existing data-flow testing techniques, as they partly overlap with path-based testing techniques. In addition, existing algorithms resolving problems similar to the current research question, in particular (4) search-based software engineering methods providing specific solutions to path-based testing problems, (5) ant-colony optimization techniques in MBT, (6) previous employment of genetic algorithms in MBT, (7) other nature-inspired algorithms employed in similar cases, and (8) alternative approaches to reliability testing of IoT systems under conditions of weak network coverage.

These eight directions of relevant research literature are discussed in the following subsections.

2.1 Model-based Testing and Used Models

MBT is based on the definition of the SUT model and the subsequent generation of test cases from this model [11], [12]. These test cases must satisfy a specified test coverage criterion, which is a set of rules determining the properties of the test cases. A model is created for a selected part or aspect of the SUT. As MBT can significantly improve the effectiveness of the entire system development process [17], it has been extensively researched in recent decades [18], [19] and is widely used in industry [20]. Moreover, its application scenarios are extremely diverse, for instance, automobiles, cell phones, and web-app development [21]–[23]. Furthermore, MBT can be applied across various types and levels of testing, such as unit testing [24], system testing [25], integration testing [26], and regression testing [27].

The selection of a specific MBT technique depends on the test strategy and character-

istics of the SUT, which correspond to a specific modeling notation [18]. As the method proposed in this thesis is based on the SUT process model, which is an extension of a directed graph, path-based testing techniques [11], [12] and data-flow techniques [28] are natural candidates for a detailed investigation of related research.

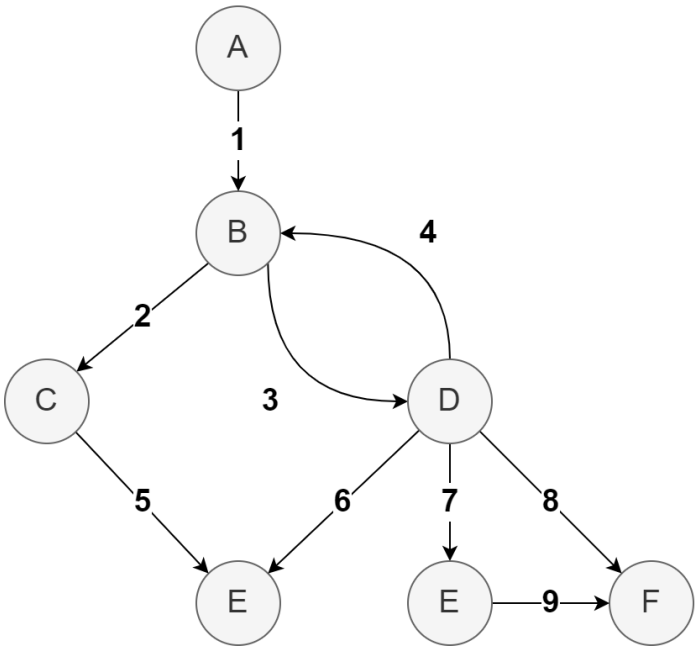
2.2 Preliminaries and the Path-based Testing

Starting with the state-of-the-art path-based testing, an established model of the problem is available, and several test-case-generating algorithms have been proposed for various test coverage criteria [13], [29]–[33]. In principle, the general concept of the path-based testing problem model is based on a directed graph $\mathcal{G} = (N, E, n_s, N_e)$, where N denotes a nonempty finite set of nodes, $E \subseteq N \times N$ is a finite set of edges, n_s denotes a start node of $\mathcal{G}$ with no incoming edges, and N_e denotes a set of end nodes of $\mathcal{G}$ with no outgoing edges [11], [12], [31]. For modeling an SUT, the literature offers two options based on the representation of the process actions and decision points:

- A function, activity, an action, or a decision point in a process is captured by $n \in N$ and the transition between two functions, activities, actions, or decision points by $e \in E$ [12].
- Only the decision points in a process are captured by the nodes N. Individual functions, activities, or actions between the decision points, including the sequences of the functions, activities, or actions uninterrupted by any decision point, are modeled by edges E [34].

A test case is typically defined as a path from n_s to any node $n_e \in N_e$. A set of test cases must satisfy a defined test coverage criterion. From the most common criteria, the *Edge*, *Edge-pair*, and *Prime path* coverage can be mentioned [12], [31]. To satisfy the *Edge* coverage, each edge of $\mathcal{G}$ must be present at least once in at least one test case t in a set of test cases T [12]. To satisfy *Edge-pair* coverage, each possible combination of the two adjacent edges in $\mathcal{G}$ must be present at least once in at least one $t \in T$ [12]. To satisfy *Prime path* coverage, each prime path in $\mathcal{G}$ must be a subpath of a test case $t \in T$. A path between two nodes of $\mathcal{G}$ is considered a prime path, if is a simple path (no inner node appears more than once in the path) and it does not appear as a proper subpath of any other simple path [12].

In general, individual algorithms support both modeling alternatives [13], [30]–[33]. As individual algorithms differ in their ability to provide the best solution for satisfying the given test coverage criteria, combining them to a portfolio strategy is a practical alternative for the testing practitioner [31].



Test Requirements satisfying Edge-pair coverage
1. [1-2]
2. [1-3]
3. [2-5]
4. [3-4]
5. [3-6]
6. [3-7]
7. [3-8]
8. [4-2]
9. [4-3]
10. [7-9]

Test Paths
1. [1-2-5]
2. [1-3-4-2-5]
3. [1-3-4-3-6]
4. [1-3-7-9]
5. [1-3-8]

Figure 2.1: Example of an SUT problem model, a set of test requirements, and a resulted set of test paths.

Considering the principle and objective of the proposed technique explained in Section 4, the current concepts that facilitate the touring of defined sequences of two non-adjacent edges (or nodes) in a path must be analyzed. The fundamental concept involves the **set of test requirements**, which are paths in $\mathcal{G}$ that must all be present as subpaths in the test cases [12], [29]. An example directed graph of a SUT processes, a set of test requirements that satisfies the Edge-pair coverage, and a resulted set of test paths is depicted in Figure 2.1.

Regarding the complexity of a general path-based testing problem, it depends on the selected test coverage criterion and the type of cost for which the set of test cases is minimized. Various options exist for defining the cost of the test cases. We can (1) minimize the size of the set of test cases (in terms of the number of test cases) that leads to a polynomial (P) time complexity - this problem is in a P class of computation problems. Nonetheless, the other common options are significantly more complex, resulting in nondeterministic polynomial-time complete (NP-complete) complexity, such as (2) minimizing the total number of nodes in the test set, (3) maximizing the ratio of the number of test requirements in the test paths, and (4) bounding the ratio of the number of test requirements in the test paths [35]. Therefore, proposing novel algorithms to solve these problems and comparing their effectiveness with those of existing algorithms is an important research direction.

As for the existing algorithms that create test cases, if not explicitly designed to satisfy

a particular test coverage criterion, they typically accept test requirements as one of their inputs [13]. These test requirements are a concept that can be partially but not completely applied to solve the current research problem that is the subject of this thesis.

To model the current problem via test requirements assuming that the network connection is interrupted and restored in only one part in the tested process, we can define a set of test requirements $R = \{r_{interrupted}, r_{restored}\}$, both of which is one edge long. In particular, $r_{interrupted}$ models an SUT function in which the network connectivity is interrupted, and $r_{restored}$ models an SUT function in which the network connectivity is restored. In this example, $r_{interrupted}$ must be followed by $r_{restored}$ in the test case, ensuring that the path from n_s to any node of N_e is minimal.

Although certain algorithms accept R as an input work to optimize the final test set, there is no assurance that $r_{interrupted}$ will be followed by $r_{restored}$. To the best of our knowledge, no published algorithm accepted additional constraints that define the order of the input test requirements. This is because there is no motivation for such functionality in standard path-based testing, and the algorithm generating the test set would become unnecessarily complex.

However, such a functionality is essential for the problem described in Section 4.

Partially, the concept of the test requirements can be applied if, in the discussed example, $R = \{$a path from $r_{interrupted}$ to $r_{restored}$ $\}$. The algorithms that accept R can be employed [13] and must be accompanied by an additional algorithm to prepare such a set of test requirements, which we applied in this study to develop the Test Requirements-based algorithm, one of the baselines for the comparative analysis of the proposed algorithms (detailed in Section 5.2).

Considering that more "$r_{interrupted}$ and $r_{restored}$" pairs can be present in the model, it is practically undecidable if such an approach yields the best solution. This uncertainty motivates the exploration of new alternative algorithms that provide the best solution to a defined problem, as we do in this thesis.

2.3 Data-flow Testing

Data-flow Testing (DFT) is another field that provides relevant information. This field overlaps with the general path-based testing, as discussed in Section 2.2. Generally, data flow testing is not classified as a subset of path-based testing, because the DFT problem can be solved by alternative means, for instance, employing a Create, Read, Update, Delete (CRUD) matrix as an SUT model [34], [36]. Nonetheless, these two fields include a certain degree of overlap.

As the DFT principle has features similar to the problem solved in this thesis, this

field is a natural candidate for detailed analysis. In this analysis, we focus on a relevant subpart of DFT based on directed-graph-based SUT models.

The DFT concept was introduced by Herman [37] in 1976 and has been thoroughly studied in recent decades. Recently, Su *et al.* [28] summarized the advantages and limitations of DFT and analyzed three types of data flow tests. The first type is called static and is based on static analysis and its search for patterns of data anomalies; the second type is dynamic, which locates invalid data usage during program execution [38]; the third option, hybrid, combines the previous two principles [28]. Notably, dynamic DFT is relevant to our thesis and its main principle is to verify the variables of the SUT by inspecting their definition and use, which is accomplished by extracting the *def-use pairs* from the code. Each pair is tested according to the selected test coverage criterion [28]. Although numerous test coverage criteria can be screened for selection [39], prior research suggests that the most effective criterion is the all-uses criterion that covers every definition and use associations in the program at least once [40]. The dynamic DFT process involves the (1) construction of the program's Control Flow Graph (CFG), (2) identification of relevant paths in the CFG that satisfy the given coverage criterion, and (3) test data generation to execute the set of paths [41].

A CFG is a directed graph constructed from the source code of an SUT containing nodes and edges. The nodes in the CFG represent a block (linear sequence) of program instructions with a single entry point (the first instruction executed) and an exit point (the last instruction executed). The edges connecting the nodes in the CFG represent the transitions between the control blocks in the control flow of a program during execution [42].

Although the CFG is used as the underlying model of the problem in DFT, which differs from the problem model specified in Section 4.1, the principle of finding paths with specific pairs of nodes in the CFG to ensure their presence in test cases is similar to that of sequencing $r_{interrupted}$ and $r_{restored}$ in the SUT model. However, for the coverage criteria specified in Section 4.2, the path between $r_{interrupted}$ and $r_{restored}$ must be directed to lead inside the SUT model part affected by limited network connectivity, i.e., not leaving it by any node other than that present in $r_{restored}$. This contributes to the novelty of the proposal presented in this thesis and its difference from previous techniques in the DFT field.

2.4 Analysis of Existing Algorithms and Strategies

We deliberately analyzed the existing algorithms and strategies in a separate section because, in certain cases, the contexts of path-based testing and DFT inevitably overlap.

In contrast to previous sections that provide the overall context and explain the problem models including their possible limitations, here we focus on the principle of the strategies and algorithms in more detail.

In relation to the principles of these algorithms, search-based and nature-inspired algorithms are two of the relevant streams analyzed in this thesis, as these are the traditional sources for pertinent studies in path-based testing. Generally, search-based and nature-inspired algorithms have to be understood as conceptually different categories [43], [44]. However, an algorithm or strategy can be search-based and also belong to nature-inspired algorithms.

2.4.1 Search-based Software Engineering Strategies

Several approaches exist for locating test paths in models, and Su classifies them into five main groups [28]. We further inspect the most researched one, the *search-based* approach that involves the use of a metaheuristic. Some other categories from the Su classification are, for example, a *collateral coverage* principle, which optimizes test set generation when covering multiple test objectives by the individual test cases, and a *random testing-based* approach that locates the test cases in the input set at random [28].

Generally, metaheuristic search-based optimization techniques used to solve general combinatorial problems in software engineering are referred to as a *Search-based Software Engineering (SBSE)* [43], [45]. In the testing domain, the field is called *Search-based Software Testing (SBST)*, and its techniques are employed for combinatorial and path-based testing [46]. The SBSE and SBST methods look for solutions in an extremely large search space with numerous constraints and competing and conflicting objectives. Therefore, an appropriate fitness function is often used to guide the search for the best solution [47]. Several SBST techniques are relevant to our research. These techniques generate paths in the graphs, modeling the SUT processes [29], [48].

Harman describes the concrete methods to perform the test paths selection [49]. He divides the techniques into two categories: *classic techniques*, such as linear programming and the branch and bound method; and a *metaheuristic search*, which consists, for example, of hill climbing, simulated annealing, and genetic algorithms methods. Another class of algorithms (that find solutions in vast search spaces) is a *nature-inspired algorithms* class, simulating, for example, the behavior of ants [50], bees [51], fireflies [52], particle swarms [53], and even microorganisms [30]. Recently, Dey *et al.* described this class of algorithms in great detail [54].

The concrete implementation of the individual SBSE and SBST methods has been thoroughly studied in recent decades. Due to its straightforwardness and high effectivity, the most researched method in SBSE is probably the genetic algorithm [55]–[59].

SBST provides several approaches that could be used to solve the problem presented in this thesis. Hence, we have selected two SBST methods, utilizing ant colony optimization and genetic algorithm, and combined them with two other algorithms using different principles (employing the breadth-first search-based graph traversal principle and extension of an existing set-covering algorithm) to create a well-balanced portfolio that would yield the best results for any given problem instance.

2.4.2 The Ant Colony Optimization Approach

Marco Dorigo introduced the Ant Colony Optimization (ACO) algorithm [60], [61] in his Ph.D. thesis in the early 90s. Inspired by the behavior of ants while they are searching for the shortest path to their food source, it became an effective instrument in solving many graph traversal-related nondeterministic polynomial (NP) problems. In the ACO implementation, *ants* are individual agents that traverse a search space between specific start and target points. The traversal is guided by the combination of *pheromone disposal* and *desirability*, information connected to the edges of a graph being traversed. Each ant leaves a pheromone trace on every edge it visits, which evaporates after some time. Therefore, the shorter the path between the source and target points, the stronger the intensity of the pheromone trace. The desirability value determines the quality of a given edge, and therefore, its formulation depends on the problem definition. It can represent, for example, a distance (number of edges) between the points for the shortest-path-related problems or a heuristic value for the traveling salesman problem [62].

The use of the ACO approach for test case generation was proposed recently in several studies. An example of this is one that was carried out by Srivastava *et al.*, modeling an SUT by a CFG [63]. In this proposal, the ants traverse the SUT model from a start node to one of the end nodes, led by a combination of pheromone disposal (the quantity of pheromones left on the edges) and a heuristic value on the edges, prioritizing those not visited yet. Their method satisfies all-path coverage in the generated set of paths [63].

To generate test sequences effectively, taking into account the importance of covering the most critical states, Srivastava *et al.* use Markov chain-based statistical MBT as well as the ACO algorithm [50]. In this method, the SUT model contains probabilities of transitions between individual application states, from which test engineers can effectively generate test cases that match their testing priorities. Altogether, this proposed method gives good coverage of critical nodes with a small number of test sequences [50]. A similar technique is also proposed by Sayyari and Emadi [64].

Furthermore, Ghiduk proposes using the ACO algorithm for the DFT. He uses ACO not only to search a CFG for paths that satisfy all-path coverage but also to generate a suite of test data that activate those paths in the CFG [65].

To summarize the related work in this field, we describe our findings using our SUT model elements, as explained in Section 4.1. Although the analyzed techniques offer the possibility of defining the critical states that must be present in T, which could potentially be used in our defined model to cover the Limited Connectivity Zone border nodes, these techniques do not ensure the sequence of nodes in the test cases, as our concept requires. By using analyzed previous techniques, it is not certain that an Limited Connectivity Zone IN node $n_{in} \in in(G, threshold)$ precedes an Limited Connectivity Zone OUT node $n_{out} \in out(G, threshold)$ in zone $L \in \mathcal{L}(G, threshold)$. Therefore, we would not be able to guarantee satisfying the selected test coverage criterion (see Section 4.2) by previous techniques. Our proposed approach satisfies these test coverage criteria, making it a novel contribution among established path-based testing techniques, including those that employ the ACO approach.

2.4.3 The Genetic Algorithm Approach

John Holland settled the basic principles of the Genetic Algorithm (GA) in 1975 [66], and they were further analyzed and described in many studies that followed [67]–[69]. The GA is a heuristic learning model based on natural evolution and selective breeding principles. This model consists of a *population* of structures, called *chromosomes*, which represent candidate solutions to a given problem; the *selection mechanism*, which chooses the best members of the population for reproduction, primarily by evaluating the *fitness function* of each chromosome; and some *genetic operators*, which slightly alter the *genes* in chromosomes in order to create new chromosomes [68].

For the purpose of software testing or system testing in general, there are two main application areas of the GA principle: (1) test data generation, and, (2) test paths generation.

Several studies describe the use of the GA to generate test data [70]–[73]. In this approach, each chromosome represents a selection of values of all input parameters of the SUT. The GA then generates a population of chromosomes that satisfies the selected test coverage criterion [55].

The employment of the GA for test path creation is the most relevant variant to our research. The individual papers differ in the use of different underlying models of the SUT (for the white-box testing of the source code, they use the CFG [74], [75], and for the black-box testing of the SUT processes, they use the Unified Modeling Language (UML) diagrams [76]). Additionally, they differ in the test coverage criteria that the generated test paths satisfy.

In path-based testing, Ghiduk uses GA to automatically generate basis test paths from a graph-based SUT control-flow model [74]. Girgis and Ghiduk make a similar

proposal in this area [75]. How the GA can be implemented to generate paths in a test program is explained in other studies [77]–[79]. At higher levels of testing, GA has been employed by Sharma *et al.* to generate test paths in UML activity diagrams modeling SUT processes and workflows. This proposal allows for utilizing activity diagrams with parallelism constructs (fork and join) [80].

In their overview study, Hermandi *et al.* discuss the possible limitations of path-based testing and challenges of using GA in this field from a more high-level point of view [56].

To effectively achieve Prime-path coverage of a sequence diagram modelling a SUT, GA is used by Hoseini and Jalili [76]. The principle of the proposed algorithm works as follows: firstly, a sequence diagram of an SUT is transformed into a control-flow graph (which is a structure that is, in this study, different to established CFG in DFT); secondly, prime paths are found in the control-flow graph; lastly, the GA is used to generate an optimal set of test paths from these prime paths [76]. Even though the results of this work are promising, generated test paths fulfill the prime path test coverage criterion, which is unnecessarily strong criterion to solve the problem addressed by our thesis.

Another use of the GA is for basis path testing, a powerful structural testing approach that employs a vector space and its basis to construct test paths [74]. Ghiduk introduces a technique that uses a GA to generate a set of test paths for basis path testing [74]. The proposed GA begins with only entry and exit edges in an initial population. The technique evaluates each chromosome by a fitness function value and selects parents of the next generation by a roulette wheel method. The crossover operator and mutation are applied afterward to the individuals chosen for reproduction. A breeding phase follows, in which an adjacent edge extends each chromosome to represent a complete path in the future. The last proposed genetic operator is an elitist operator that enhances the offspring of the current generation by its best member. The algorithm terminates when a produced set of individuals (paths through a CFG) represents a basis set of paths. The basis set of paths must consists of independent paths (those paths that never appear as sub-paths of any other path in the CFG), the set of paths must contain all edges in the CFG, and every path not contained in the basis set of paths must be constructible by a linear combination of the paths in this set [74].

Girgis *et al.* propose their version of the GA that produces test sets that satisfy the all-uses criterion [75] for DFT. The selection method uses the roulette wheel method, and when the SUT code contains loops, the proposed algorithm searches for paths using the ZOT-subset criterion: "Each loop in a program is iterated zero, one, and two times in execution" [81].

To summarize, in the area of the GA applied in path-based and data-flow testing, we have not identified a proposal that would address the limited network connectivity

problem, as is the subject of this thesis, for the same reason that we explained in the last paragraph of Section 2.4.3.

2.4.4 Other Nature-inspired Approaches

A variety of additional algorithms for solving combinatorial and path-based testing problems also find inspiration from nature. Apart from the ACO and GA discussed in the sections above, another example from a general class of swarm intelligence algorithms, is the Partical Swarm Optimization (PSO) algorithm. Generally, in the PSO principle, the search space exploration imitates the behavior of swarms, such as schools of fish or flocks of birds. Windisch *et al.* propose how to perform structural testing using the PSO and state that this technique is much simpler, easier to implement, and has fewer parameters that the user has to adjust than with the GA [53].

Another nature-inspired algorithm adapted for the automated generation of test paths is the Artificial Bee Colony (ABC) search algorithm. It consists of three types of bees, the *scout* bees, which search randomly for new food sources; the *onlooker* bees, which decide which food source to process based on information from the last category of bees; the *employed* bees, which fly to the food sources and determine nectar amounts in these sources [82]. Lam *et al.* use the independent and parallel behavior of all types of bees mentioned earlier to satisfy all independent test path coverage criteria [51]. Compared to other nature-inspired algorithms, they state that the ABC-inspired algorithm finds the solution more quickly.

Even fireflies, light-emitting flying bugs, inspired researchers to simulate their behavior to solve optimization problems. Yang proposed a firefly algorithm based on the following: all fireflies are attracted to each other, their attractiveness is proportional to the brightness of the light they emit (those with a duller light will move toward those with a brighter light, and those with an equal level of brightness will move randomly), and the brightness of a firefly is determined by the objective function [83]. Therefore, during each round of the algorithm, all fireflies move from their initial location through the search space toward the brighter ones, storing the best solution (the location of the brightest firefly) in each round. Srivatsava *et al.* use the FA approach to guide the graph traversal to create paths that represent test cases in CFGs or state-based models of an SUT [84]. The authors extend the original approach, defined by Yang [83], with the assumption that the fireflies lose their intensity of brightness as they move through the graph (instead of using space absorption coefficient that was employed in the previous versions of the algorithm) and that the distance between fireflies is computed as the sum of edges between nodes, instead of the original Cartesian distance in the space. The test path generation starts by calculating the input model's objective function, which is followed by the generation

of fireflies in each of the model's nodes. The fireflies then traverse the graph led by the guidance factor (calculated using cyclomatic complexity and the graph adjacency matrix), looking for the best path. The fireflies can then further prioritize the order of the paths in the test set by calculating the mean of the brightness of each path.

Internal mechanisms of the slime mold Physarum Polycephalum, a large single-celled amoeboid organism, is used by Arora *et al.* [30] as inspiration to generate test scenarios for the concurrent sections in the UML activity diagram. Using statistical analysis, the authors demonstrate that the proposed Amoeboid Organism Algorithm approach is better than the existing ACO and GA approaches. This choice is made because of the redundancies in those algorithms when traversing the search space, leading to savings in the time needed to generate the set of test cases. On the other hand, there aren't any findings on the quality of the generated paths through the models in the paper. Also, it is unclear what leads the agent through the traversal of the search space. Hence, it is difficult to evaluate how this method can be used in relation to the subject of this thesis.

While analyzing all the path-based testing algorithms mentioned in this section, we were not able to identify any that directly addressed the problem we introduce in Section 4 and define formally in Section 4.1.

Algorithms that can be utilized to solve our problem are those that accept the SUT model G we define in this thesis (see Section 4.1), or its variant, based on a directed graph and a set of test requirements. Some proposals are given by Li *et al.* [13], namely Brute force, Prefix-graph-based, or Set Covering algorithms. In this thesis, we utilize the Set Covering algorithm and accompany it with a particular procedure to prepare test requirements, as we present in Section 5.2.

2.5 Alternative Techniques for Testing IoT System Functionality with a Limited Network Connectivity

The research community approaches the testing of IoT systems from several perspectives. From the test organization perspective, Tan and Cheng propose a division according to test levels: unit tests, integration tests, system tests, and acceptance tests [85]. Another perspective present Murad *et al.*, who divide it into usability tests, reliability and scalability tests, compatibility tests, security tests, data integrity tests, and performance tests [86]. Because these test levels are inspired by the widely researched area of software testing, they often lack IoT specificity [85], [86].

However, there are some exceptions in the literature. As well as an overview of existing tools for testing the IoT systems, Dias *et al.* introduce the IoT-specific division of testing

activities into the edge testing, fog testing, and cloud testing categories. From their perspective, edge testing covers testing the low-level parts of the IoT system. At the same time, fog testing spans the middle layer of the IoT system, network connection, and security. Lastly, cloud testing addresses the cloud perspective of the IoT system, meaning the scalability and dynamic configuration [87].

Focusing on network connectivity testing specifically, Muthiah and Venkatasubramanian introduce the term "connectivity testing" for this purpose [88]. The need to perform these connectivity tests is mentioned, for example, by Murad *et al.* in their example from the healthcare industry [86]. The same term is used by Sirshar *et al.* in their preprint about software quality assurance testing methodologies in IoT [89]. Additionally, Esquiagola *et al.* perform connectivity tests on their IoT platform [90].

Although insufficient attention has been paid to limited network connectivity testing focusing on processes in IoT systems, some alternatives exist. The alternatives mainly focus on lower levels of an SUT, typically on a network level [91]–[93]. The most frequently explored topic in existing studies is Quality of Service (QoS) testing [5], [92], [94], [95].

In 2017, White *et al.* analyzed 162 research articles to carry out a systematic mapping study on state-of-the-art QoS approaches in IoT [92]. The study found that the most researched layers of the IoT infrastructure were the Open Systems Interconnection (OSI) model's physical, link, and network layers up to that year. On the other hand, the deployment, middleware, and cloud layers lacked further research. However, this research concentrated on individual layers of the IoT system infrastructure and didn't consider the high-level process viewpoint of the system.

Another study in this direction was done by Rudeš *et al.* to present a concrete example of QoS assurance for IoT systems [91]. The study involved the testing of a small sensor network prototype that shares its data, over the internet, with a server located in a laboratory. However, the tests were aimed only at the network communication quality and not the influence on the overall process. Matz *et al.* provide an analysis of quality assurance for network communication between IoT systems on the physical and application layers [94]. The authors measured the quality of a Narrowband-IoT technology that provides energy-efficient and long-range network access to IoT devices using the cellular network, e.g., Long Term Evolution (LTE) or 5G in the future. Kim *et al.* propose a service-based automated IoT testing framework to resolve constraints regarding coordination, costs, and scalability issues of traditional software testing [96]. This framework also performs remote distributed interoperability testing, scalable and automated conformance testing, and semantic testing. However, the set of test cases that run on the SUT is predefined, making the exhaustive process of testing under a limited network connectivity impossible [96].

Analyzed studies in this area typically assess network reliability and related topics. Higher levels of SUT functionality (functional correctness from the system user's viewpoint or flawless integration) are not tested. Hence, the techniques for testing IoT functionality are underexplored from the perspective of system behavior. No specific path-based, data-flow, or SBST technique directly addressed the given goals of this thesis.

A systematic mapping study on the aspects of quality assurance in IoT systems that our lab recently conducted also confirms this conclusion [5]. The study found that one of the areas not sufficiently covered in the literature and, therefore, worth exploring is the development of specific test design techniques to test IoT systems with a limited network connection, which is the issue addressed by this thesis.

2.6 Summary of Related Work and Motivation

Although there is no unified definition of which types of systems the IoT family includes, all these systems employ the Internet (or a closed data network) as the connecting element essential for the functionality of the individual components of a given system. Upon examining the use cases of various IoT systems, wireless networks were primarily applied to connect these components. Owing to its nature, a wireless network can experience connection outages. This occurs if a mobile device is used in a location with problematic network coverage (e.g., uninhabited areas, tunnels, subways), or in case of an energy shortage in the infrastructure, or defects or damage in the hardware components.

In all such systems and primarily those applied in the critical infrastructure, manufacturers should ensure the appropriate functioning of the system, even in case of a network outage and restoration. As analyzed and explained in this section, this research area requires further attention.

There are several reasons to approach the limited network connectivity tests from a process perspective, as we do in this thesis. Firstly, owing to the potential model sizes and versions of the individual components of the IoT system that can be combined, the number of test cases can be enormous [97]. Second, the testers of smart devices should test these devices in the real world outside the lab, where Internet connectivity may be intermittent [98]. Any test performed in this manner requires considerable time and resources, which favors the MBT approach because it can automatically generate a precisely optimized set of test cases containing only the most significant test cases. Third, the testing process of IoT systems should be automated to reduce testing costs and enable faster execution of these tests [99], [100]. Thus, the process perspective provides a highly insightful basis for this approach.

To model the processes of an IoT system, the initial models (e.g., UML activity di-

agrams) can be transformed into directed-graph-based models with specific properties. A test case is a path through this graph-based model, and as discussed in Section 2.2, path-based testing techniques are generally employed to generate these types of test cases.

However, the existing path-based testing methods do not address the specificity of the testing problem described in Section 4.1. This problem model contains several Limited Connectivity Zone (LCZ)s, each of which represents a subsystem undergoing network disruption. This LCZ is connected to the stable components of the IoT system through the LCZ IN and LCZ OUT nodes (defined in Section 4.1), whose pairs must be present in the test cases. More precisely, to visit the LCZ IN and OUT node pair on the borders of LCZ z using a test case t, the LCZ IN node must be placed before the LCZ OUT node in t. Furthermore, the path between the LCZ IN and LCZ OUT nodes in t must not exit z. This rule is further described in Section 4.2, where we formalize the test coverage criteria.

This specific condition was not observed in the studied path-based testing techniques. The only exception in the field that can be utilized is the *test requirement* concept, which was employed in this project. As explained further in Section 5.2, we transformed *EachBorderOnce* and *AllBorderCombinations* coverage criteria (defined in Section 4.2) to test these requirements and used the algorithm proposed by Li *et al.* [13] to generate a set of test cases satisfying these test requirements.

However, this is only one possible approach. In this thesis, we explored more alternatives that can outperform an approach based on the *test requirement* concept.

In addition to the analysis of the existing literature, we should consider an industrial perspective for motivation. Accordingly, we received positive feedback from several companies with which our lab has been cooperating during the last four years, namely, Skoda Auto, Rockwell, Siemens, and Electrolux. After explaining the principle of the technique, all testing specialists from these companies confirmed that the proposed approach is reasonable from an industrial tester's viewpoint and encouraged us to pursue the development and implementation of this technique.

Chapter 3

Thesis Statement and Research Questions

The research statement for this PhD project is as follows:

For a specific problem of testing the functionality of an IoT system under limited network connectivity, the current established path-based testing techniques do not suffice, and thus, a specialized technique has to be developed. Such a technique will produce a set of test cases effective in terms of the testing effort required to identify relevant defects in the system under test.

The research questions for the PhD project are inquired as follows:

- **RQ 1:** How to model the problem of testing the functionality of an IoT system under limited network connectivity such that a path-based testing approach is utilized?
- **RQ 2:** Which existing approaches and algorithms can be utilized or partially utilized to solve the defined problem?
- **RQ 3:** Which new algorithms developed specially for the discussed problem can generate a set of test cases that is effective in the sense of the testing effort required to find relevant defects in a system under test?
- **RQ 4:** Considering the nature of the path-based testing discipline, is it possible to formulate the singularly best-performing algorithm, or would a strategy based on a combination of several algorithms deliver the best solution?

In this thesis, **RQ 1** is addressed in Sections 4.1 and 4.2. The proposed model is further verified through a set of experiments, as described in Section 8. **RQ 2** is answered

by analyzing the existing algorithms reviewed in Section 2 and summarized in Section 2.6. The development and proposal of an algorithm based on the established test requirements concept is elaborated in Section 5.2. Notably, this algorithm serves as a baseline for comparison with the proposed algorithms to obtain the first answer of **RQ 3**. Thereafter, three additional algorithms are proposed in Sections 4.4.1, 4.4.2, and 4.4.3, which were combined in the portfolio strategy presented in Section 6. The algorithms were executed on the set of SUT models described in Section 7 and evaluated using criteria introduced in Sections 4.3, 5.1, and 7.3. The experimental results are discussed in Section 8 and further discussed in Section 9. Finally, based on the experimental results, an answer to **RQ 4** is provided in Section 9.

Chapter 4

Proposed Limited Network Connectivity Technique

Important requirements for the testing process are to perform it cost-optimally and with a sufficient level of abstraction on concrete subsystems, protocols, and platforms. Therefore, to satisfy these requirements, we chose to concentrate on the process viewpoint of IoT systems behaving under a limited network connectivity, and we propose a process testing technique, called LNCT.

It is a black-box testing technique, and its output is a set of test cases that represent end-to-end paths through the SUT processes (hence path-based testing). Such a technique is useful during system testing in the later phases of the system development life cycle. But, it can be used even in integration or regression testing of the SUT.

To verify the functionality of an IoT system affected by limited network connectivity or connectivity outages, test designers must construct a set of test scenarios addressing this problem. In this thesis, the test design is primarily focused on testing the following two principal situations:

1) In a particular part of a process handled by the SUT, the network connectivity is interrupted (or limited to an extent that affects the functionality of the SUT). In such a case, functional testing should be conducted on the SUT for the following scenarios[1]:

1. When a subsystem of the SUT is isolated from network connectivity for a certain period as it collects data, are these data transmitted and correctly stored offline until the network connectivity is restored, or do the collected data become irretrievable?

2. A SUT subsystem accepts signals, e.g., commands or application programming interface (API) calls, from other devices or subsystems, and this receiving subsystem

[1]The given situations are only examples, and the list is not exhaustive but may include certain test situations that are not relevant to all types of IoT systems

is temporarily disconnected from the network. Are the other signal-transmitting devices notified regarding the missing (offline) subsystem failing to respond to these signals?

3. Is a SUT user notified regarding the limited functionality caused by network connectivity outage?

2) The network connectivity is restored after an outage. At the instant the network connectivity is recovered, the following typical situations require testing:

1. If the SUT data must be processed transactionally, is this transactionality maintained even in a network connectivity outage? Specifically, will the affected transactions be discarded in a deterministic manner or completed via the available caches upon recovering the connection? Are the cached transactions finished correctly, including the logical order of their steps?

2. When SUT devices, modules, or subsystems cache the data during the connectivity outage, are these cached data accurately transmitted to the receiving SUT modules after the network connectivity is restored? Although this transmission might affect the responsiveness or performance of the SUT, is such a temporary fallout acceptable for the users and system safety?

3. Are the data stored and processed by the SUT consistent (their internal structure and content not malformed) after network restoration and transmission of the locally stored data or closure of the transactions?

4. Is a user of the SUT notified intently that the erstwhile disabled functionality is available again?

In this thesis, we approached this problem from the perspective of process (or path-based) testing. We aimed to execute a process flow in an SUT to learn the behavior of the process as it is affected by a network connectivity outage or limitation. To test the outlined situations, we must construct path-based test cases in which we follow the events when the network connectivity is interrupted (or goes to be limited) by the events in which the connectivity is restored. An exemplary case is illustrated in Figure 4.1.

A sample fictional IoT system composed of three subsystems (devices and backend systems) is presented in Figure 4.1, wherein Subsystems A and C are IoT devices: Subsystem A is connected to a stable network, whereas Subsystem C is a mobile device operating in an area under limited network connectivity (e.g., rural, maritime, or subterranean areas). Subsystem B is the backend connected to a stable network.

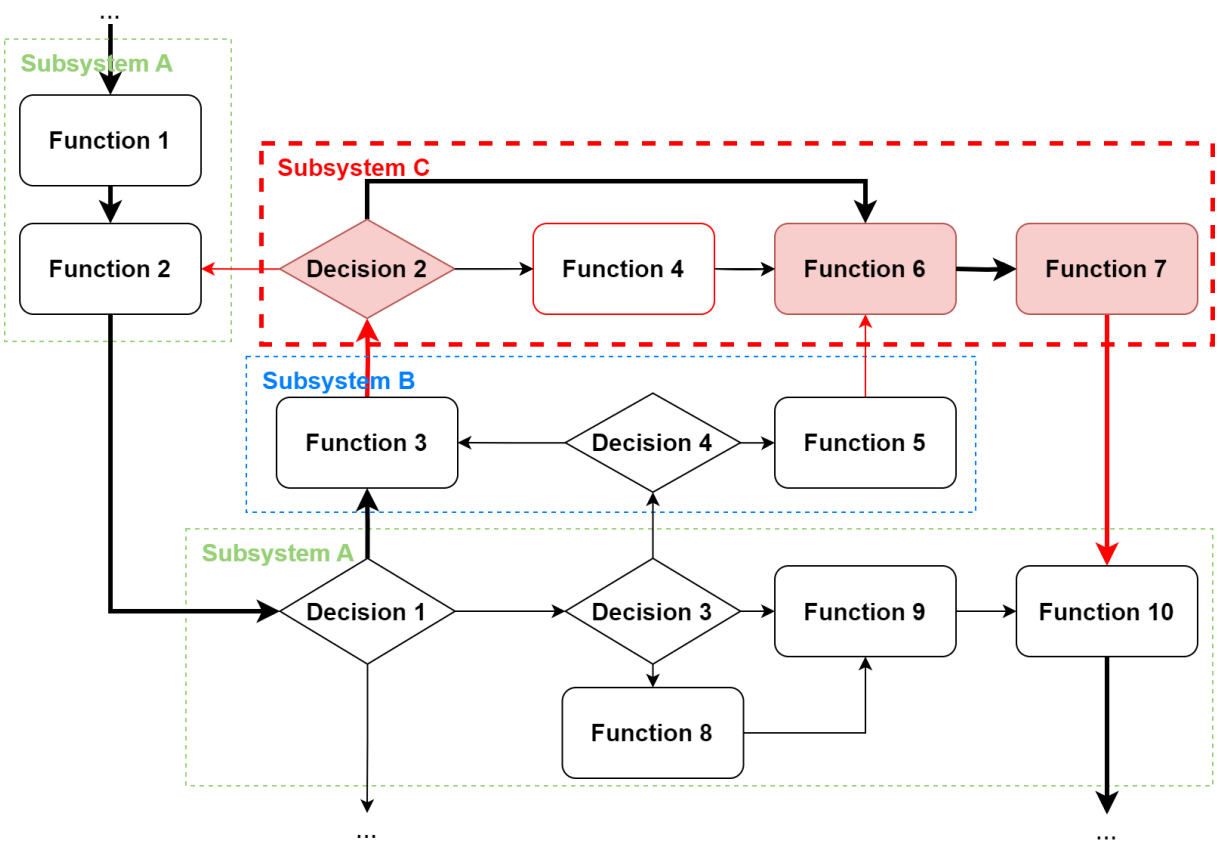


Figure 4.1: Initial example of an SUT affected by a network connectivity outage in its submodule and a path-based test case enabling the test of the SUT behavior in such situations.

In the test, we assumed that Subsystem C operates even without network connectivity. We used various process flow variants to learn the system behavior under such restrictions. In the current example, Functions 4, 6, and 7, and Decision 2 are affected by network connectivity outages and depicted with the red border. To test the outlined situation, we exercised a transition from Function 3 to Decision 2, Decision 2 to Function 2, Function 5 to Function 6, and Function 7 to Function 10 (these transitions are depicted in red in Figure 4.1). In the test scenario presented in Figure 4.1 (bold arrows) as an example, the event is sequenced till the network connectivity is interrupted (or limited, transitioning from Function 3 to Decision 2) along with the record of the event in which connectivity is restored (transitioning from Function 7 to Function 10).

Using an SUT model, various test paths sequencing the events of network connectivity outage with the events of connectivity restoration can be identified. However, a significant number of test cases would not be optimal from the perspective of the overall testing expenditure. Thus, our goal is to generate cost-effective test sequences to address the limited network connectivity problem.

4.1 Model of the Problem

The **SUT process**, which can be affected impacted by a possible network connection outage (CO), for which we create the test cases, is abstracted as a directed graph $G = (N, E, n_s, N_e)$, where $N \neq \emptyset$ represents a finite set of nodes, and E denotes a nonempty set of directed edges, $E = \{(n, m) \mid (n, m) \in N \times N$ is an ordered pair of nodes $\}$. The node $n_s \in N$ is the initial/start node of the graph G (with no incoming edges), and N_e defines a nonempty set of end nodes of graph G, $N_e = \{n_e \mid n_e \in N$ has no outgoing edge$\}$. In particular, the nodes serve as abstractions of the SUT actions, functions, or decision points, and the edges represent the transitions between them in the process flow. In the proposed method, G does not permit parallel edges.

Test case t is a sequence of nodes $n_1, n_2, ..., n_n$, with a sequence of edges $e_1, e_2, ..., e_{n-1}$, where $e_i = (n_i, n_{i+1})$, $e_i \in E$. Test case t starts with start node n_s ($n_1 = n_s$) and ends with end node ($n_n \in N_e$). Test set T is a set of test cases. Alternatively, we used the term *test path* for the test case in the text.

Edge connection outage probability (COP) denoted by $cop(e)$ is defined for $e \in E$ and indicates a percentage representing the abstracted probability of a connection outage in this edge. If particular value of $cop(e)$ is defined, then e is a transition affected by the possible limited network connectivity in the IoT system.

Threshold COP denoted by $threshold$ represents the threshold connection outage probability, for which the test set T is created. By setting $threshold$ to n, we assume that all edges with a COP greater than or equal to n will be affected by a hypothetical network connectivity outage.

Furthermore, we introduce the concept of LCZ. **LCZ edge** is an edge $e \in E$ for which $cop(e) \geq threshold$ and **Non-LCZ edge** denotes an edge $e \in E$ with $cop(e) < threshold$. **LCZ** L represents a coherent subgraph of G containing only the LCZ edges. For a given $threshold$, G can contain more than one LCZ. LCZs of G are denoted by $\mathcal{L}(G, threshold)$.

IN node of LCZ L denotes a node n satisfying one of the following conditions:

1. $n = n_s$ and n has an outgoing edge that is an LCZ edge of L.
2. n has an outgoing edge that is an edge of L, and n has an incoming edge that is not an LCZ edge of L.

$in(L) \subset N$ denotes all IN nodes of L and $in(G, threshold) \subset N$ denotes all IN nodes of all LCZs $\mathcal{L}(G, threshold)$.

OUT node of LCZ L is a node n that satisfies one of the following conditions:

1. $n \in N_e$ and n has an incoming edge that is an edge of L.

2. n has an incoming edge that is an LCZ edge of L, and n has an outgoing edge that is not an LCZ edge of L.

$out(L) \subset N$ denotes all OUT nodes of L and $out(G, threshold) \subset N$ denotes all OUT nodes of all LCZs $\mathcal{L}(G, threshold)$.

Border node of LCZ is either the IN or OUT node of LCZ

Using the fictional IoT system outlined in Figure 4.1 as an example, these concepts are illustrated in Figure 4.2. In the example, $N_e = \{n_{11}, n_{16}\}$, L contains n_5, n_6, n_7, and n_8, $in(L) = \{n_5, n_7\}$ and $out(L) = \{n_5, n_8\}$. The sample test case can be $t = n_s, n_1, n_2, n_3, n_4, n_5, n_7, n_8, n_{15}, n_{16}$.

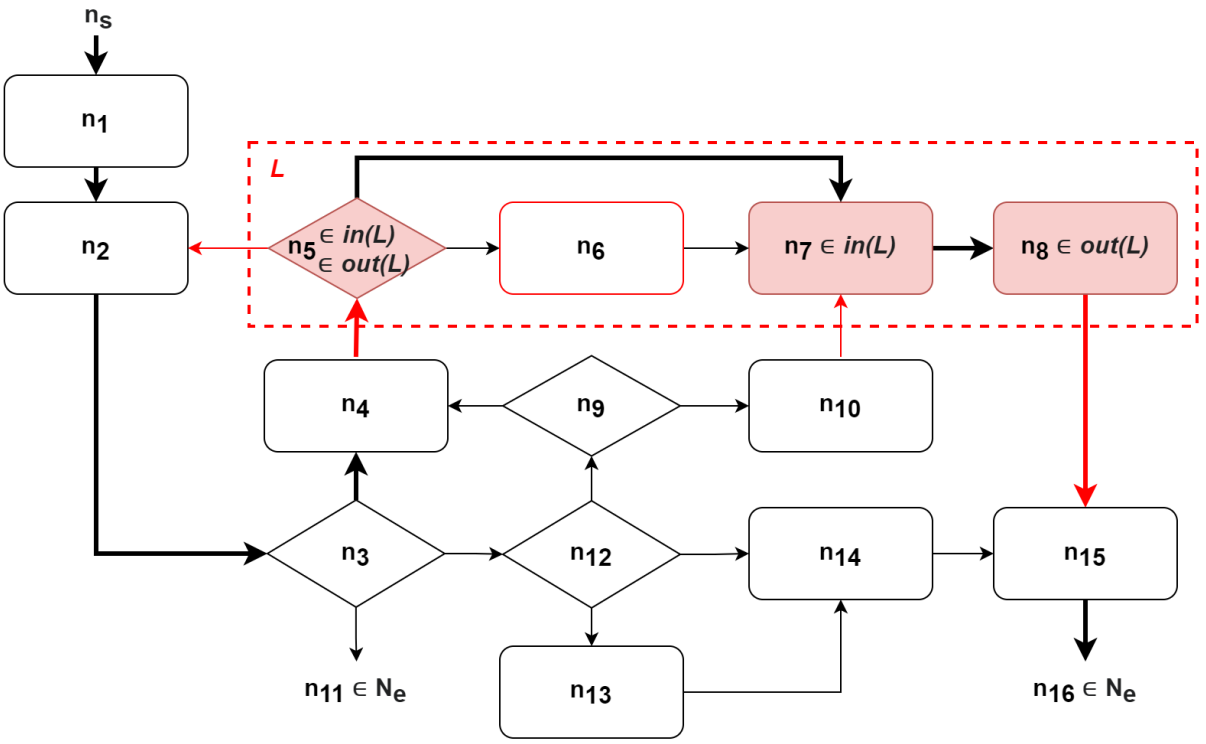


Figure 4.2: Illustration of defined model elements in exemplary IoT system.

The **test case generation problem** is stated as follows: Given the SUT model G, *threshold*, and test coverage criterion $\mathcal{C}$, determine a test set T satisfying $\mathcal{C}$. The test coverage criteria are described in Section 4.2.

4.1.1 Problem Model Transformation Guidelines

We summarize the transformation process of the original SUT model to the format we propose in the following list of steps. On the input of the transformation is SUT process model, *threshold* value, and test coverage criterion $\mathcal{C}$.

1. Locate the start node in the process model; if there is no dedicated start node, create a new node and make it enter all nodes that have no incoming edges.

2. Make all the nodes in the process model reach a certain end node. If there is a node that doesn't reach it, create a new node and insert it into the model to the place where the modeled process is terminated.

3. Make all transitions in the process model to be directed edges from a single node to a different node. If there are loops in the graph, transform them with the possible help of the insertion of some artificial nodes.

4. If there are parallel edges in the graph, transform them. It is possible to insert an artificial node between the start and end nodes of the parallel transitions.

5. Estimate the connection outage probabilities of all the transitions in the SUT model and insert this value to the dedicated edge in the model.

6. Based on the connection outage probabilities of the transitions, locate the IN nodes and OUT nodes in the model by following the conditions defined in the previous section. The limited connectivity zones, comprised of the IN nodes, OUT nodes, and inner nodes (for which all the incoming and outgoing edges have $cop(e) \geq threshold$), are then located in the model.

After the last step of the transformation, the model contains all the information and is in the correct state to be inputted together with $threshold$ and $\mathcal{C}$ to the algorithms we propose in this thesis for the test set generation.

4.1.2 Problem Model Transformation Example

We demonstrate the model transformation process on the following system that we cooperate on. It is a sensor network aimed to reduce fatal casualties in defense operations, called Digital Triage Assistant (DTA). In the original high-level diagram depicted in Figure 4.3, we model the basic function of the DTA system in the defense operation, during which soldiers wear it, and it constantly communicates with the command center and shows the health status, position, and other data regarding the soldiers.

Individual processes are identified and modeled in the following stages of the analysis. A model of such a process is depicted in Figure 4.4.

We then transform the process model into the model using the elements defined in this chapter. Based on the analysis, we assign the COP levels, which are then used to identify the LCZs and border nodes. In Figure 4.5, we present the visualization of such a model; in this case, it is the transformed model from Figure 4.4. In the figure, there are two LCZs visualized by brown transitions of the LCZ edges, nodes with orange backgrounds of the border nodes, and nodes with white backgrounds and brown circles of the LCZ inner nodes.

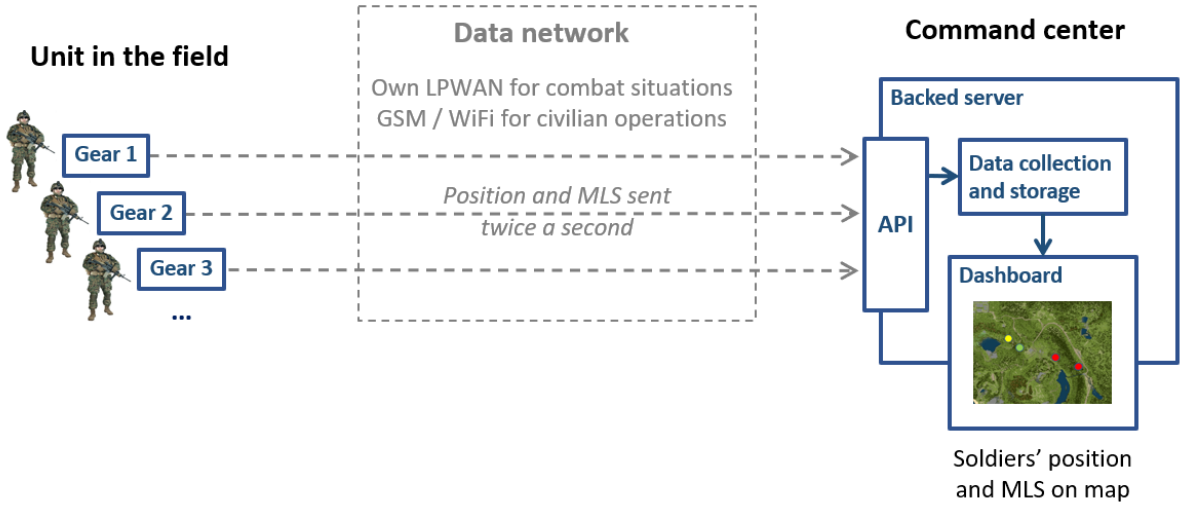


Figure 4.3: High-level view at the DTA functionality in a defense operation.

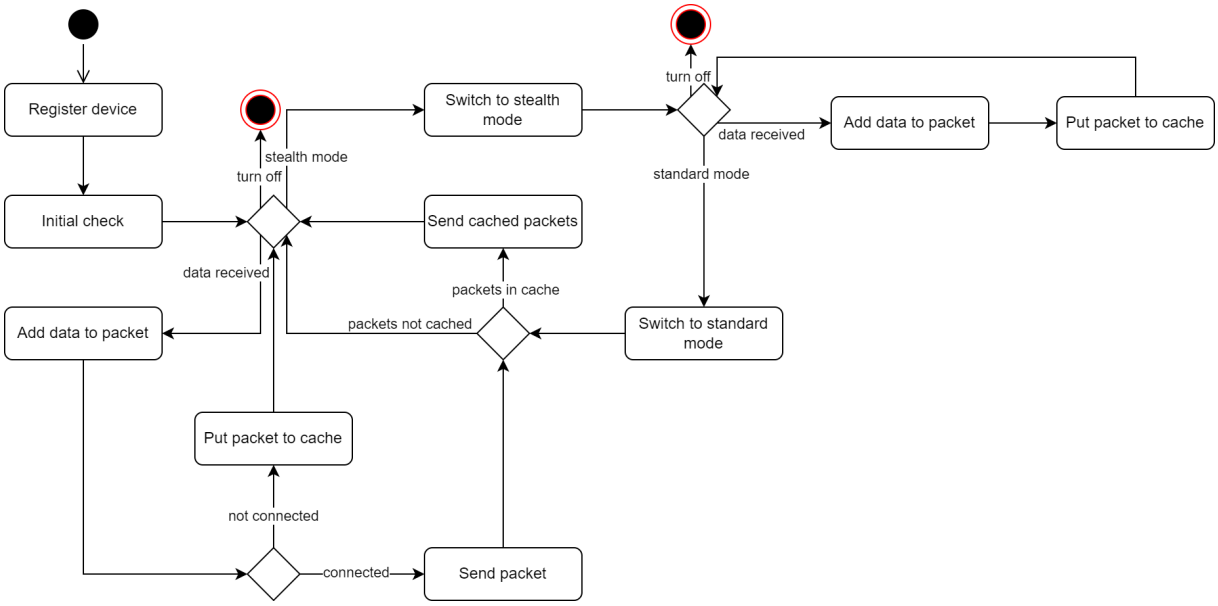


Figure 4.4: UML diagram of DTA with stealth mode process.

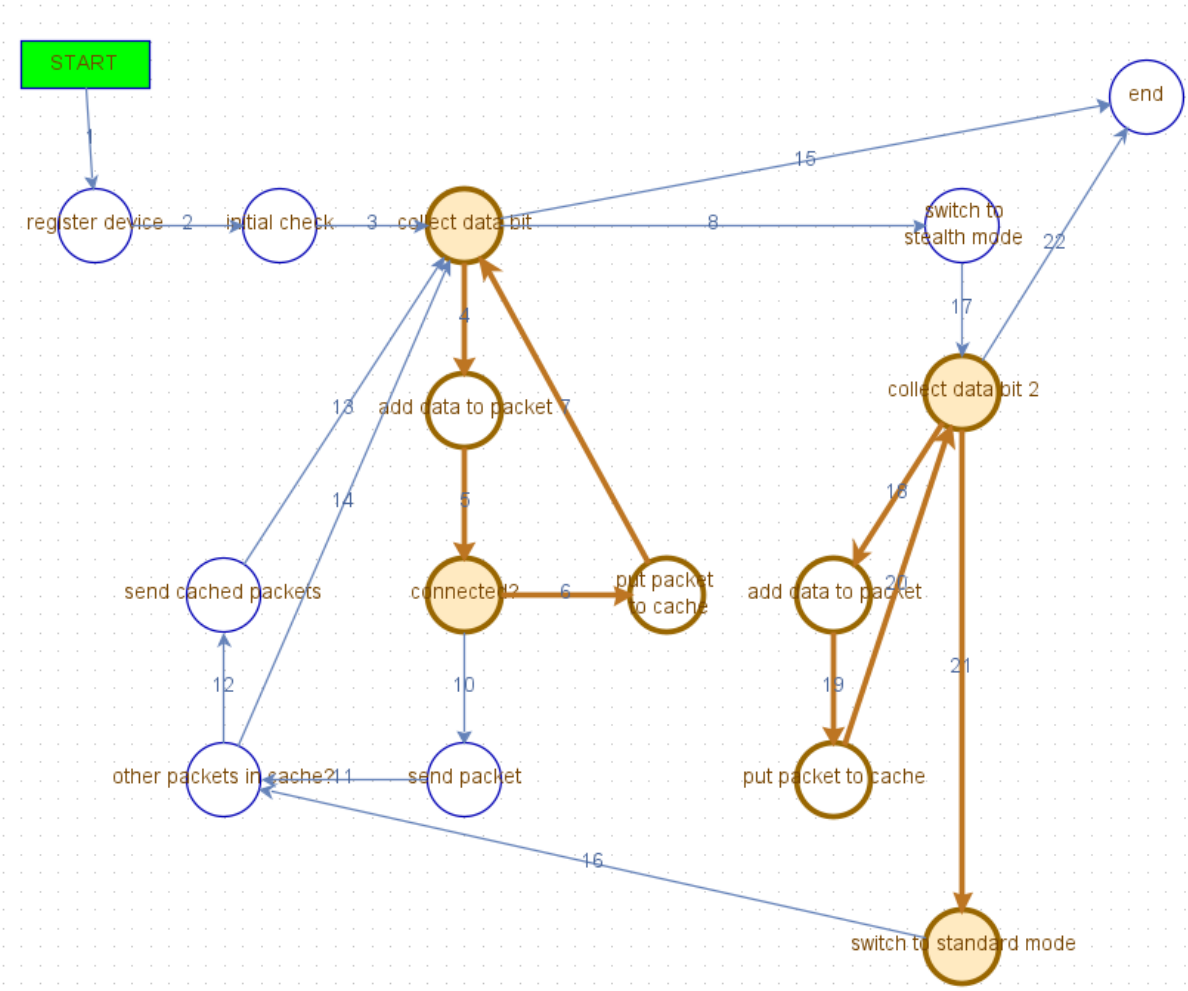


Figure 4.5: An LNCT example problem model of a DTA with stealth mode process.

4.2 Test Coverage Criteria

For the limited connectivity problem discussed in this thesis, several test coverage criteria $\mathcal{C}$ can be defined, and here, we defined four test coverage criteria: *EachBorderOnce*, *AllBorderCombinations*, *ComprehensiveEachBorderOnce*, and *ComprehensiveAllBorderCombinations*. These test coverage criteria differ according to the number of test case steps and the method of test case construction.

To satisfy the ***EachBorderOnce*** criterion, the test set T must contain each node of $in(G, threshold)$ and $out(G, threshold)$. Furthermore, for all $L \in \mathcal{L}(G, threshold)$, if $t \in T$ contains a node $n_{in} \in in(L)$, then this node must be followed by a node $n_{out} \in out(L)$ later in the test path t but not necessarily immediately. The proposed technique allows n_{in} to be equal to n_{out}, because in certain situations, the LCZ may be entered and exited through the same node. Practically, this test coverage criterion requires that the test cases visit all the LCZ border nodes at least once, regardless of the path for entering the IN and exiting the OUT nodes.

To satisfy the ***AllBorderCombinations*** criterion, for each $L \in \mathcal{L}(G, threshold)$, the test set T must contain each combination of a node $n_{in} \in in(L)$ and a node from $n_{out} \in out(L)$, for which a path exists from n_{in} to n_{out} inside the L. Furthermore, for all $L \in \mathcal{L}(G, threshold)$, if $t \in T$ contains a node $n_{in} \in in(L)$, then this node must be followed by node $n_{out} \in out(L)$ later in the test path t, but not necessarily immediately, to satisfy this coverage criterion. Note that n_{in} can be equal to n_{out}. Informally, this test coverage criterion requires that the test cases must traverse through all possible combinations of the IN and OUT nodes of the LCZ borders for which there exists a path inside the LCZ from the IN node to an OUT node, regardless of the edges followed to enter the IN and exit the OUT nodes in the test case.

To visit the nodes susceptible to network outage using more possible combinations of paths beyond the LCZs, we define two additional test coverage criteria suitable for testing the more critical components of the SUT processes.

To satisfy ***ComprehensiveEachBorderOnce***, T must satisfy *EachBorderOnce*, and the following conditions must be satisfied: each $n_{in} \in in(G, threshold)$ must be entered by a Non-LCZ edge in a $t \in T$ and each $n_{out} \in out(G, threshold)$ must be exited by a Non-LCZ edge in a $t \in T$.

To satisfy the ***ComprehensiveAllBorderCombinations***, for each $L \in \mathcal{L}(G, threshold)$, the test set T must contain all combinations of a node $n_{in} \in in(L)$ and a node from $n_{out} \in out(L)$ for which a path exists from n_{in} to n_{out} within L. Furthermore, n_{in} must be entered by a Non-LCZ edge in a $t \in T$ and n_{out} must be exited by a Non-LCZ edge in this t. Furthermore, for all $L \in \mathcal{L}(G, threshold)$, if $t \in T$ contains node $n_{in} \in in(L)$, then this node must be (later in the test path t but not necessarily immediately) followed by

node $n_{out} \in out(L)$ to satisfy this coverage criterion. Note that n_{in} can be equal to n_{out}.

These novel test coverage criteria are applicable in various situations based on the criticality of the SUT and the available testing resources. Informally, the *EachBorderOnce* criterion is the weakest and results in the lowest number of test steps in T. The *AllBorderCombinations* coverage criterion cover all possible combinations of the LCZ border nodes. Therefore, it induces more rigorous testing and a greater number of test steps in T. The "Comprehensive" variant of both criteria, *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations*, further increase the potential number of test steps in T, as they require exiting an LCZ before visiting another LCZ IN node in a test case.

4.3 Test Set Evaluation Criteria

To evaluate the test set cost or quality, several test-set evaluation criteria have been discussed in the literature for path-based testing [13], [101], [102]. For the current research problem discussed in this thesis, the options defined in Table 4.1 were employed as test set evaluation criteria $\mathcal{E}$.

Evaluation criterion	**Description**
$\|T\|$	Number of test cases in test set T.
$l(T) = \sum_{i=1}^{\|T\|} \|t_i\|, t_i \in T$	Total length of test set T measured in number of edges.
$s(T) = \sqrt{\frac{\sum_{i=1}^{\|T\|}(\|t_i\| - \overline{\|t\|})^2}{\|T\| - 1}}, \|T\| > 1$	Length dispersion of the test cases in the test set T, expressed by a standard deviation of test case lengths; the test case length is measured in the number of edges.
$U(T) = \frac{u\_edges(T)}{l(T)} \cdot 100\%$	Ratio of unique edges in test set T to the total number of edges in test set T, where *u_ edges(T)* denotes the number of unique edges in test set T.
$B(T) = \frac{b\_nodes(T)}{l(T) + \|T\|} \cdot 100\%$	Ratio of the number of border nodes in test set T to the total number of nodes in test set T (which is $l(T)+\|T\|$), where *b_ nodes(T)* denotes the number of border nodes in test set T for all LCZs of G.

Table 4.1: Test case evaluation criteria $\mathcal{E}$.

From these criteria, the total length of the test set T, $l(T)$ served as a proxy for the

effort required to conduct the tests. The count of test cases in a test set T, $|T|$, is an auxiliary indicator for evaluating the results of the individual algorithms. A lower value of $l(T)$ (for T satisfying the given test coverage criterion) indicated more effective test cases from the standpoint of testing effort. Although $|T|$ can be interpreted in the same manner, $|T|$ must be considered in the context of $l(T)$ and an isolated value of $|T|$ does not directly express the effort required to execute the tests.

The length dispersion of a test set T ($s(T)$) is vital for preventing excessively long and short test cases. Generally, test analysts consider lengthy test cases impractical because the probability that the test case is interrupted by a defect and the remainder of the test cases cannot be completed increases with its length. Consequently, as a higher $s(T)$ implies higher variability in the length of the test case, a lower value of $s(T)$ implies a more suitable test set.

Criterion $U(T)$ provides additional information on the effectiveness of the test cases, and its goal is to express a (potentially ineffective) repetition of G edges in the test cases. A higher $U(T)$ value indicates better T.

Similarly, $B(T)$ expresses the effectiveness of the test set for visiting the LCZ border nodes in relation to the total number of G nodes visited during the tests, i.e., a higher value of $B(T)$ indicates better T.

In the present experiments, we applied the defined $\mathcal{E}$ to evaluate the properties of T generated by the algorithms for various SUT models.

4.4 Proposed Algorithms

In this thesis, we present three algorithms that generate T from G for the LNCT technique:

1. **Shortest Paths Composition (SPC)** is based on the principle of determining the shortest path between the nodes identified in G and connecting them in the test cases (described in Section 4.4.1);

2. **Ant Colony Optimization-based algorithm (ANT)** employs the ant-colony optimization principle to find the test cases (described in Section 4.4.2);

3. **Adapted Genetic Algorithm-based paths generation (AGA)** employs the principle of genetic algorithm to derive the test cases (described in Section 4.4.3).

4.4.1 Shortest Paths Composition Algorithm

The main routine of the SPC is described in Algorithm 1, which accepts G, *threshold*, and $\mathcal{C}$ as inputs and produces T as its output, while maintaining a set of unused LCZ IN

nodes U_{in} and a set of unused LCZ OUT nodes U_{out}. The Algorithm 1 starts by finding all shortest paths between $in(L)$ and $out(L)$ inside all LCZ $L \in \mathcal{L}(G, threshold)$ of given G. They are generated by a subroutine $FindPathsInLCZs$ and stored to P.

The subroutine $FindPathsInLCZs$, described in Algorithm 2, works on the breadth-first search principle starting in $out(L)$ nodes of each LCZ $L \in \mathcal{L}(G, threshold)$. Nodes to traverse are stored in queue Q. For all explored nodes, the distance from a particular node $n_{out} \in out(L)$ is stored. In the algorithm, this distance is denoted as $distance(n_1, n_2)$. Then, for each L, paths from each $in(L)$ with the shortest distance to $out(L)$ are selected as an output, denoted as P. In Algorithm 2, $parents(n)$ denotes the set of parents of node n.

The main routine of Algorithm 1 continues by exploring G using the breadth-first search principle. When this search reaches a start node of any path $p \in P$, the exploration history stored in $potential\_previous\_TC\_step$ is expanded by nodes in p, and the end node of p is added to the queue Q that contain nodes, from which further exploration of the graph is conducted.

The selection of path $p \in P$, which is going to be used in the currently constructed test case, is specified in procedure $GetNextShortestPathInLCZ$ (Algorithm 3). This procedure also manages the removal of p from P according to the selected coverage criterion $\mathcal{C}$. For *ComprehensiveAllBorderCombinations* this procedure just removes p from P. For *AllBorderCombinations* or *EachBorderOnce*, this procedure removes all paths $p' \in P$ that are sub-paths of p. For the *EachBorderOnce* and *ComprehensiveEachBorderOnce* test coverage criteria, all paths p', ending with already-used LCZ OUT nodes (nodes that are not in U_{out}), are removed from P.

Then, the exploration of G continues searching for another start node of different path $q \in P$. When an end node of G is reached during the exploration, a test case is composed based on the exploration history and added to a set of test cases T that the SPC routine returns as its result.

Algorithm 1: $SPC(G, threshold, \mathcal{C})$**:** Combine shortest paths between the start node and some of the end nodes through the the LCZs in the test cases.

Input : SUT model G, $threshold$, and coverage criteria $\mathcal{C}$

Output: set of test cases T

```
1  T ← ∅; U_in ← in(G, threshold); U_out ← out(G, threshold)
2  P ← FindPathsInLCZs(G, threshold) ;   ▷ ∀p ∈ P must be contained in T
3  while P ≠ ∅ do
4      PUT n_s ∈ G to Q ;                ▷ Q is a queue of nodes to traverse
5      set path as empty ;               ▷ path is a sequence of consecutive nodes
6      while Q is not empty do
7          n ← POP from Q ;              ▷ n is a currently traversed node
8          if n ∈ in(G) and P contains a path that starts in n then
9              p ← GetNextShortestPathInLCZ(C, P, n, U_in, U_out)
10             P ← P \ {p}
11             o ← the last node of p ;          ▷ o ∈ out(G, threshold)
12             for each n_p ∈ p do
13                 potential_previous_TC_step(n_p) ← parent(n_p) ; ▷ Save parent
                    - child connection
14             end
15             SET Q as empty ;                  ▷ Delete all elements in Q
16             PUT o to Q
17         end
18         else if n ∈ N_e then
19             t is a new path of nodes; Add n to t
20             temp ← n
21             while potential_previous_TC_step(temp) has been set do
22                 add potential_previous_TC_step(temp) at the beginning of t
23                 temp ← potential_previous_TC_step(temp)
24             end
25             T ← T ∪ t
26         end
27         else
28             for each d ∈ descendants(n) do
29                 potential_previous_TC_step(d) ← n
30             end
31         end
32     end
33 end
34 return T
```

Algorithm 2: *FindPathsInLCZs*$(G, threshold)$: Find all relevant shortest paths inside LCZs present in the SUT model G.

Input : SUT model G, *threshold*

Output: set of shortest paths between $in(L)$ and $out(L)$ inside the LCZ L for all $L \in \mathcal{L}(G, threshold)$, denoted as P

```
1  P ← ∅
2  for each LCZ L ∈ 𝓛(G, threshold) do
3      for each n_out ∈ out(L) do
4          SET Q as empty ;                ▷ Q is a queue of nodes to traverse
5          PUT n_out to Q
6          for each x in L except n_out do
7              distance(n_out, x) ← ∞ ;    ▷ distance equals to number of nodes
                of a path from n_out to x
8          end
9          while Q is not empty do
10             n ← POP from Q
11             for each p ∈ parents(n) do
12                 if distance(n_out, p) > distance(n_out, n) then
13                     distance(n_out, p) ← distance(n_out, n) + 1
14                     PUT p to Q
15                 end
16             end
17         end
18         for each n_in ∈ in(L) do
19             SET path as empty ; ▷ path is a sequence of consecutive nodes
20             n ← n_in ;                  ▷ n is a currently traversed node
21             while n ≠ n_out do
22                 n ← x ∈ L such that distance(n, x) is minimal
23                 ADD n at the end of path
24             end
25             ADD n at the end of path
26             if |path| > 1 then
27                 P ← P ∪ path
28             end
29         end
30     end
31 end
32 return P
```

Algorithm 3: $GetNextShortestPathInLCZ(\mathcal{C}, P, n_{in}, U_{in}, U_{out})$**:** Return the shortest path inside LCZ from n_{in} with respect to given coverage criterion $\mathcal{C}$.

Input : a coverage criteria $\mathcal{C}$, set of shortest paths P, an LCZ IN node n_{in}, set of unused LCZ IN nodes U_{in}, and set of unused LCZ OUT nodes U_{out}

Output: next shortest path p

$n_{out} \leftarrow$ a node to which exists a path from n_{in} present in P
if $\mathcal{C} = EachBorderOnce$ ***or*** $\mathcal{C} = ComprehensiveEachBorderOnce$ **then**
  if $n_{out} \notin U_{out}$ ***and*** P *contains a path that ends in an* $n'_{out} \in U_{out}$ **then**
    $n_{out} \leftarrow n'_{out}$
  end
end
$p \leftarrow$ a path from n_{in} to n_{out} that is present in P
$P \leftarrow P \setminus \{p\}$
if $\mathcal{C} = AllBorderCombinations$ ***or*** $\mathcal{C} = EachBorderOnce$ **then**
  for *each* $n \in p$ **do**
    for *each path* p' *from* n *that is present in* P **do**
      $n'_{out} \leftarrow$ the end node of path p'
      if n'_{out} *follows* n *in path* p **then**
        $P \leftarrow P \setminus \{p'\}$
      end
    end
  end
end
if $\mathcal{C} = EachBorderOnce$ ***or*** $\mathcal{C} = ComprehensiveEachBorderOnce$ **then**
  $P \leftarrow P \setminus \{\, p \mid p \in P$ and p starts in n_{in} and leads to an $n_x \notin U_{out} \,\}$
end
return p

4.4.2 Ant Colony Optimization-based Algorithm

The ANT algorithm is inspired by the Ant Colony Optimization (ACO) algorithm that we already introduced in Section 2.4.2. The main routine of ANT is described in Algorithm 4, which accepts G, *threshold*, and $\mathcal{C}$ as inputs and produces T as its output. The following section presents the algorithm variables, their initiation, the algorithm steps, and procedures, e.g., how to obtain the desirability and pheromone levels to guide the traversal of G and how to choose the ant that found the best path.

Constant	Value
α	1
β	3
NC	15
ρ	0.5
m	30
c	1.0

Table 4.2: The values of the constants for the ANT algorithm.

Algorithm Variables and Constants

There are two important variables that influence the ant's traversal of G. Namely, τ_{ij} represents the pheromone intensity of edge $(i, j) \in E$, and $H[(i, j)]$ stores the desirability of edge $(i, j) \in E$.

The following variables and constants are employed in the ANT algorithm:

- α: A constant representing the weight of the pheromone level in the calculation.
- β: A constant representing the weight of the desirability level in the calculation.
- NC: A constant representing the number of repetitions of an ant's search for a path.
- ρ: A coefficient representing the level of pheromone evaporation after each iteration of the ant's search for a path.
- m: A constant representing the number of ants used for the graph exploration.
- τ_{ij}: The pheromone intensity of edge $(i, j) \in E$.
- $H[(i, j)]$: The desirability of edge $(i, j) \in E$.
- c: The initial level of the τ_{ij} variable.

The values of the constants specified in Table 4.2 were found after extensive fine-tuning during the experiments, which yielded the best results for the ANT algorithm.

Algorithm Initiation

The ANT's main routine is specified in Algorithm 4. It accepts G, *threshold*, and $\mathcal{C}$ as inputs and produces T as its output. In the initial step, the algorithm creates a mapping $\mathcal{U}$ that contains a set of reachable LCZ OUT nodes $U \subset out(L)$ for every node $n \subset in(L)$. This mapping is constructed for all LCZs L. In the Algorithm 4, *LCZ OUT node* $r \subset U$ *reachable from node* n means that a path from n to r exists. The method for generating $\mathcal{U}$

traverses all $L \in \mathcal{L}(G, threshold)$ that begin in $n_{in} \in in(L)$ nodes and end in $n_{out} \in out(L)$ nodes, using the breadth-first search (BFS) algorithm.

The *ANT* routine continues by calling the *ANTCore* procedure (see Algorithm 5) repeatedly. Among its outputs, it produces a path P that is stored into a set of test cases T, and LCZ border node pairs stored in B_{in} and $\mathcal{B}_{out}$ that are then removed from the mapping $\mathcal{U}$. Algorithm 4 continues until the mapping $\mathcal{U}$ is not empty.

In the ANT algorithm, the term *nodes covered by ants* means that a particular ant traversed a pair of LCZ IN and LCZ OUT nodes in a sequence corresponding to selected test coverage criterion $\mathcal{C}$ (see Section 4.2).

Algorithm 4: $ANT(G, threshold, \mathcal{C})$: The main routine of the ANT Algorithm that explores G and maintains a mapping of covered LCZ border nodes $\mathcal{U}$.

Input : SUT model G, *threshold*, coverage criterion $\mathcal{C}$,
Output: Set of test cases T

1 $\mathcal{U}$ is a mapping where a key is an LCZ IN node $k \in in(G, threshold)$ and a value is a set of LCZ OUT nodes from $out(G, threshold)$ that are reachable from k in G. In $\mathcal{U}$, an empty set can be stored for a particular key.
2 Initiate $\mathcal{U}$ for G and *threshold*.
3 $T \leftarrow \emptyset$
4 **while** $\mathcal{U}$ *is not empty* **do**
5 $(P, B_{in}, \mathcal{B}_{out}) \leftarrow ANTCore(G, threshold, \mathcal{C}, \mathcal{U})$; ▷ Find the best path P and covered LCZ border nodes B_{in} and $\mathcal{B}_{out}$
6 $T \leftarrow T \cup \{P\}$; ▷ Add path P to the set of test cases
7 **for** *each* $n_{in} \in B_{in}$ **do**
8 **for** $n_{out} \in \mathcal{B}_{out}$ **do**
9 REMOVE n_{out} from $\mathcal{U}[n_{in}]$; ▷ Covered LCZ OUT node n_{out}
10 **end**
11 **end**
12 **for** *each* $n_{in} \in B_{in}$ **do**
13 **if** $\mathcal{U}[n_{in}] = \emptyset$ **then**
14 REMOVE key n_{in} from $\mathcal{U}$; ▷ Covered LCZ IN node n_{in}
15 **end**
16 **end**
17 **end**
18 **return** T

The ANTCore Algorithm

The *ANTCore* procedure, described in Algorithm 5, manages the traversal of the ants through G. The ants are led by a combination of desirabilities and pheromone disposal on the edges of G.

First, the algorithm creates the list of ants; this list has length m. Second, the mapping of desirabilities $\mathcal{D}$ is initialized. For each node $x \in N$, this mapping $\mathcal{D}$ returns which LCZ border nodes are reachable from x (the mapping $\mathcal{D}$ returns a set (R_{in}, R_{out}), where reachable LCZ IN nodes are stored in set R_{in} and reachable LCZ OUT nodes are stored in set R_{out}).

The next step is obtaining the desirability levels, specified in the $FindReachableBNs$ procedure (Algorithm 6). This procedure traverses G from each end node $n_e \in N_e$ to the start node n_s, following the edges in the reverse direction.

The *ANTCore* algorithm continues by traversing G using the $ANTTraversal$ procedure (Algorithm 7) and then selecting the best ant (champion) $a \in A$ using the $FindChampion$ procedure (Algorithm 10). $ANTTraversal$ and $FindChampion$ procedures, both described in a greater detail below, are repeated NC number of times so that the pheromone levels have a more significant impact.

The output of the *ANTCore* procedure is ant a's path P and the LCZ border nodes it covers, stored in variables B_{in} for LCZ IN nodes and $\mathcal{B}_{out}$ for LCZ OUT nodes. We define B_{in} and $\mathcal{B}_{out}$ in Section *Ant Traversal*.

Finding Reachable LCZ Border Nodes

Procedure $FindReachableBNs$ specified in Algorithm 6 traverses G from its end nodes to the start node by the depth-first search (DFS) algorithm. The algorithm is storing the LCZ IN and LCZ OUT nodes that are reachable from all $x \in N$ into $\mathcal{D}$.

Ant Traversal

Algorithm 7 defines the process of traversing the SUT model G by a selected ant. For each ant $\xi \in A$, the algorithm starts to traverse G from the node n_s. During the traversal, it stores the nodes that ant ξ visited in variable P. The LCZ border nodes that ant ξ covered according to the selected test coverage criteria $\mathcal{C}$ are stored in a set B_{in} and a mapping $\mathcal{B}_{out}$. The set B_{in} contains LCZ IN nodes. Storage of the LCZ OUT nodes $\mathcal{B}_{out}$ is implemented as mapping $\mathcal{B}_{out} : b_{in} \rightarrow B_{out}$, where $b_{in} \in in(G, threshold)$ and $B_{out} \subset out(G, threshold)$.

As introduced earlier, the traversal of G by ant ξ is guided by a combination of pheromone disposal and desirability levels on the edges. The desirability level of each

Algorithm 5: $ANTCore(G, threshold, \mathcal{C}, \mathcal{U})$: Traverse the SUT model by ants, who are trying to visit as much LCZ border nodes as possible; return the champion when finished.

Input : SUT model G, *threshold*, coverage criterion $\mathcal{C}$, a mapping of yet uncovered nodes $\mathcal{U}$

Output: The best path P found, a set of LCZ IN nodes B_{in} covered in P, and a mapping of LCZ OUT nodes $\mathcal{B}_{out}$.

Set all constants for the ANT algorithm described in 4.4.2 to the values specified in Table 4.2.
A is a set of m ants
$\mathcal{P}$ is a mapping of ant paths, where particular ant ξ is the key and its path is the value $\mathcal{P}[\xi]$
$\mathcal{D} \leftarrow \emptyset$; $\mathcal{D}$ is a mapping of a node $n \in G$ to a set (R_{in}, R_{out}), $R_{in} \subset N \in G$, $R_{out} \subset N \in G$, where nodes from R_{in} are reachable from n and nodes from R_{out} are reachable from n. This mapping is created for all $N \in G$ and $\mathcal{D}[x]$ denotes particular (R^x_{in}, R^x_{out}) for a node x.
for *each* $n_e \in N_e$ **do**
    $V \leftarrow \emptyset$; ▷ `A set to store visited nodes`
    $\mathcal{D}' \leftarrow FindReachableBNs(G, threshold, \mathcal{U}, \mathcal{D}, V, n_e)$
    $\mathcal{D} \leftarrow \mathcal{D}'$
end
$count \leftarrow 0$
while $count \leq NC$ **do**
    $count \leftarrow count + 1$
    for *each* $\xi \in A$ **do**
        Place ant ξ to the position of node n_s
        $(P', B'_{in}, \mathcal{B}'_{out}) \leftarrow ANTTraversal(G, threshold, \mathcal{C}, \mathcal{U}, \xi)$
        $\mathcal{P}[\xi] \leftarrow P'$, $\mathcal{N}_{in}[\xi] \leftarrow B'_{in}$, $\mathcal{N}_{out}[\xi] \leftarrow \mathcal{B}'_{out}$
    end
    $a \leftarrow FindChampion(A, \mathcal{N}_{in}, \mathcal{N}_{out}, \mathcal{P})$; ▷ `Select the best ant` $a \in A$
    for *each edge* (i, j) *in* $\mathcal{P}[a]$ **do**
        $\tau_{ij} \leftarrow \tau_{ij} + \frac{1}{|\mathcal{P}[a]|}$; ▷ `Deposit pheromone on edges in` $\mathcal{P}[a]$`, for` τ_{ij} `see Section 4.4.2,` $|\mathcal{P}[a]|$ `denotes the number of nodes in` $\mathcal{P}[a]$
    end
    for *each edge* $(i, j) \in E \in G$ **do**
        $\tau_{ij} \leftarrow (1 - \rho) * \tau_{ij}$; ▷ `Pheromone decay of all edges, for` ρ `see Section 4.4.2`
    end
end
$a \leftarrow FindChampion(A, \mathcal{N}_{in}, \mathcal{N}_{out}, \mathcal{P})$; ▷ `Find the ant with the best path upon all iterations`
$P \leftarrow \mathcal{P}[a]; B_{in} \leftarrow \mathcal{N}_{in}[a]; \mathcal{B}_{out} \leftarrow \mathcal{N}_{out}[a]$
return $(P, B_{in}, \mathcal{B}_{out})$

Algorithm 6: *FindReachableBNs*(G, *threshold*, $\mathcal{U}$, $\mathcal{D}$, V, n): Traverse the SUT model G to find from which nodes we can reach which LCZ border nodes.

Input : SUT model G, *threshold*, a mapping of uncovered LCZ border nodes $\mathcal{U}$, a set of already visited nodes V, mapping $\mathcal{D}$ that for each node $x \in N \in G$ returns which LCZ border nodes can be reached from x, traversed node n

Output: Updated $\mathcal{D}$ after all recursive iterations of this subroutine

```
1  for each p ∈ parents(n) do
2      if p ∉ V then
3          V ← V ∪ p
4          (R^p_in, R^p_out) ← D[p]
5          if n ∈ in(G, threshold) and n is a key in U then
6              R^p_in ← R^p_in ∪ n
7          end
8          if n ∈ out(G, threshold) and U contains a value n for any key then
9              e is an edge from p to n
10             if cop(e) ≥ threshold then
11                 R^p_out ← R^p_out ∪ n
12             end
13         end
14         D[p] ← (R^p_in, R^p_out)
15         D' ← FindReachableBNs(G, threshold, U, D, V, p) ; ▷ Recursive call
16         D ← D'
17     end
18 end
19 return D
```

edge is calculated by the *CalculateDesirabilities* procedure (Algorithm 8, described in the following paragraph) and stored in mapping $\mathcal{H}$, which maps each edge $(i, j) \in E$ to a desirability level. The desirability levels of the edges are updated after each move the ant makes because they are dependent on the B_{in} and $\mathcal{B}_{out}$ variables, which are changed during the execution of the *ANTTraversal* procedure.

When the desirability is computed, the ant moves to a randomly selected edge of G using a probability stored in a mapping $\mathcal{E}_{N_i}$. This is calculated using the equation on line 10 of Algorithm 7, which is inspired by Dorigo's formula (1) in the study *Ant colony optimization: A new meta-heuristic* [103].

Furthermore, according to the selected test coverage $\mathcal{C}$, Algorithm 7 calls the *ManageBorderNodes* procedure, specified in Algorithm 9. This procedure returns an updated set B_{in} and a mapping $\mathcal{B}_{out}$ of covered LCZ IN and LCZ OUT nodes respectively, based on the ant's current path and the node's j nature (for instance, LCZ IN node, uncovered LCZ OUT node, or other types). We introduce this algorithm details later in this section. The procedure *ManageBorderNodes* ends when the ant moves to an end node $n_e \in N_e$.

Algorithm 7: $ANTTraversal(G, threshold, \mathcal{C}, \mathcal{U}, \xi)$: The inner procedure of the *ANTCore* algorithm, which performs the SUT model traversal by ant ξ.

Input : SUT model G, *threshold*, coverage criterion $\mathcal{C}$, a mapping of uncovered nodes $\mathcal{U}$, ant ξ

Output: path P of ant ξ, a set B_{in} of LCZ IN nodes covered by ξ, a mapping $\mathcal{B}_{out}$ of LCZ OUT nodes covered by ξ

$P \leftarrow$ empty path ; ▷ P `is a path of ant` ξ

$P_L \leftarrow$ empty path ; ▷ `A path of` ξ `inside LCZ` $L \in \mathcal{L}(G, threshold)$

$B_{in} \leftarrow \emptyset$; ▷ `LCZ IN nodes set covered by ant` ξ

$\mathcal{B}_{out} \leftarrow$ Create an empty mapping $b_{in} \rightarrow B_{out}$; $b_{in} \in in(G, threshold), B_{out} \subset out(G, threshold)$

$n_{in} \leftarrow nil$; ▷ `The last LCZ IN node reached by ant` ξ `during its path`

while ξ *has not reached* n_e **do**

  $i \leftarrow$ current position of ξ, $N_i \leftarrow$ set of i descendants
  $\mathcal{H} \leftarrow CalculateDesirabilities(G, \mathcal{U}, \mathcal{C}, i, n_{in}, P, \mathcal{D}, B_{in}, \mathcal{B}_{out})$

  $\mathcal{E}_{N_i}$ is a mapping where a value is a number $< 0, 1 >$ of probability of ant ξ moving to $j \in N_i$ via edge x incoming to j and key is x

  for *all nodes* $j \in N_i$ **do**

    $\mathcal{E}_{N_i}[(i,j)] \leftarrow \dfrac{(\tau_{ij})^{\alpha} \cdot (\mathcal{H}[(i,j)])^{\beta}}{\sum_{l \in N_i} \tau_{il} \cdot \mathcal{H}[(i,l)]}$; ▷ `Store the probability of ant` ξ `moving to node` j `by edge` (i,j)

  end

  Randomly select next node $j \in N_i$ to move ant ξ to, using edge (i,j). In this selection, probability $\mathcal{E}_{N_i}[(i,j)]$ is used.

  if $cop((i,j)) \geq threshold$ **then** Add (i,j) at the end of P_L ;

  Add (i,j) at the end of P ; Move ξ from i to j using (i,j)

  if j *is a key in* $\mathcal{U}$ **then**

    $n_{in} \leftarrow j$; ▷ `node` j `is uncovered LCZ IN node`

  end

  else if j *is anywhere in values of* $\mathcal{U}$ **then**

    $B_{in} \leftarrow B_{in} \cup \{n_{in}\}$, $\mathcal{B}_{out}[n_{in}] \leftarrow \mathcal{B}_{out}[n_{in}] \cup \{j\}$

    if $\mathcal{C} = ComprehensiveAllBorderCombinations$ ***or*** $\mathcal{C} = ComprehensiveEachBorderOnce$ **then**

      $(B'_{in}, \mathcal{B}'_{out}) \leftarrow ManageBorderNodes(n_{in}, j, B_{in}, \mathcal{B}_{out}, \mathcal{U}, \mathcal{C}, P_L)$

    end

    else

      for *each node* x *in* P_L *from the beginning of* P_L **do**

        $P'_L \leftarrow$ part of P_L starting with x

        for *each node* p' *in* P'_L *from the beginning of* P'_L **do**

          $(B'_{in}, \mathcal{B}'_{out}) \leftarrow ManageBorderNodes(x, p', B_{in}, \mathcal{B}_{out}, \mathcal{U}, \mathcal{C}, P_L)$

        end

      end

    end

  end

  $(B_{in}, \mathcal{B}_{out}) \leftarrow (B'_{in}, \mathcal{B}'_{out})$

end

return $(P, B_{in}, \mathcal{B}_{out})$

The Calculation of Edge Desirability

Edge desirabilities are computed by the *CalculateDesirabilities* procedure specified in Algorithm 8. This procedure produces a mapping $\mathcal{H}$ that for each edge $(i, j) \in E$ returns its desirability $h \in < 0, 1 >$, which corresponds to the number of uncovered LCZ border nodes reachable when using edge e. The procedure works as follows. First, the algorithm initializes the mapping $\mathcal{H}$. Then, it iterates each edge (i, j) and stores the number of reachable uncovered LCZ border nodes in variable h. The algorithm gets the set of uncovered LCZ IN nodes in R^j_{in} and the set of uncovered LCZ OUT nodes from R^j_{out}, which it gets as a tuple (R^j_{in}, R^j_{out}) from the mapping $\mathcal{D}$. These sets need to be updated based on the path the ant traversed; therefore, the algorithm creates another sets $R^j_{in}{}'$ and $R^j_{out}{}'$, empty initially. Firstly, the algorithm updates $R^j_{in}{}'$ when node j is an uncovered LCZ IN node. Secondly, it updates $R^j_{out}{}'$ when node j is an uncovered LCZ OUT node. The rest of the algorithm updates $R^j_{in}{}'$, $R^j_{out}{}'$, and h according to the current position of the ant; whether it is inside, on the border, or outside any LCZ, and according to the chosen test coverage criterion $\mathcal{C}$. Specifically, h is equal to the number of nodes in sets $R^j_{in}{}'$ and $R^j_{out}{}'$. If there is at least one such uncovered node ($h > 0$), we project h into $h' \in (0, 1)$ using a function $h' = -\frac{1}{2h} + 1$ and store it with key (i, j) into $\mathcal{H}$, which is returned by the *CalculateDesirabilities* procedure when the iteration of edges is complete.

Traversing G and Covering LCZ Border Nodes

When the ant is traversing G, the algorithm keeps sets of covered border nodes B_{in} and $\mathcal{B}_{out}$ dynamically updated. This is done by using the *ManageBorderNodes* procedure, described in Algorithm 9. This procedure returns the updated set of LCZ IN nodes B_{in} and LCZ OUT nodes $\mathcal{B}_{out}$ that the ant covered, according to the selected test coverage criterion $\mathcal{C}$. First, the *ManageBorderNodes* procedure tests whether n_{in} and n_{out} aren't covered yet by checking their presence in $\mathcal{U}$, and if not, it updates B_{in} and $\mathcal{B}_{out}$ accordingly. Then, if $\mathcal{C}$ is equal to *EachBorderOnce* or *ComprehensiveEachBorderOnce*, the algorithm iterates each uncovered LCZ IN node $u_{in} \in \mathcal{U}$ that has the LCZ OUT node n_{out} in the set $\mathcal{U}[u_{in}]$ and then updates $\mathcal{B}_{out}[u_{in}]$, where the information if the LCZ OUT n_{out} is covered by this ant is stored. LCZ IN node n_{in} is covered by ant (added to B_{in}) for *EachBorderOnce* or *ComprehensiveEachBorderOnce* test coverage criteria in all cases. For the other test coverage criteria, n_{in} is added to B_{in} only when the mapping $\mathcal{B}_{out}$ of LCZ OUT nodes (covered by the current ant in use) contains all yet uncovered LCZ OUT nodes $\mathcal{U}[u_{in}]$.

Algorithm 8: *CalculateDesirabilities*(G, $\mathcal{U}$, $\mathcal{C}$, i, n_{in}, P, $\mathcal{D}$, B_{in}, $\mathcal{B}_{out}$): Calculates desirabilities of the edges according to the number of border nodes reachable from surrounding nodes and not covered yet.

Input : SUT model G, mapping of uncovered LCZ border nodes $\mathcal{U}$, coverage criterion $\mathcal{C}$, node i, uncovered LCZ IN node n_i, path P of ant, mapping $\mathcal{D}$ that for each node $n \in N \in G$ returns which LCZ border nodes it reaches, set of covered LCZ IN nodes B_{in}, set of covered LCZ OUT nodes $\mathcal{B}_{out}$,

Output: Mapping $\mathcal{H}$ for ant-routing to neighbors of i, where a key is some edge, and a value is its desirability.

Set $\mathcal{H}$ as empty
for each edge (i, j) outgoing node i
  $h = 0$; ▷ `Number of reachable uncovered LCZ border nodes when using edge` (i, j)
  $(R^j_{in}, R^j_{out}) \leftarrow \mathcal{D}[j]$; ▷ `Symbols defined in Algorithm 5`
  $R^j_{in}{}' \leftarrow \emptyset$; ▷ `Uncovered LCZ IN nodes reachable from node` j
  $R^j_{out}{}' \leftarrow \emptyset$; ▷ `Uncovered LCZ OUT nodes reachable from node` j
  if $j \in R^j_{in}$ **then**
    $R^j_{in}{}' \leftarrow R^j_{in} \setminus B_{in}$
  end
  if $j \in R^j_{out}$ **then**
    for *each LCZ OUT node* $r \in R^j_{out}$ **do**
      if r *is not in values in mapping* $\mathcal{B}_{out}$ **then**
        $R^j_{out}{}' \leftarrow R^j_{out}{}' \cup \{r\}$
      end
    end
  end
  if (i, j) *is an LCZ edge* **then**
    L is an LCZ that contains (i, j)
    if n_{in} *is not nil* **then**
      $R^j_{out}{}' \leftarrow R^j_{out}{}' \setminus \{i\}$; ▷ `Because` i `is uncovered LCZ OUT node`
      $R^j_{in}{}' \leftarrow R^j_{in}{}' \setminus in(L)$
      $h = 1 + |R^j_{in}{}'| + |R^j_{out}{}'|$
    end
    else if *last edge of path* P *is not a LCZ edge* ***and*** $i \in \mathcal{U}$ ***and*** $i \notin B_{in}$ **then**
      $R^j_{in}{}' \leftarrow R^j_{in}{}' \setminus \{j\}$; ▷ `Because` j `is uncovered LCZ IN node`
      $h = 1 + |R^j_{in}{}'| + |R^j_{out}{}'|$
    end
    else
      $R^j_{in}{}' \leftarrow R^j_{in}{}' \setminus out(L)$
    end
  end
  ... continues on the following page

```
33
34  | ... continuation of Algorithm 8
35  | else if n_in is not nil and (i ∈ U or C = ComprehensiveEachBorderOnce or
    |  C = EachBorderOnce) then
36  |   | h = 1 + |R_in^j'| + |R_out^j'|
37  | end
38  | else if n_in is nil and i ∈ U and i ∉ B_in then
39  |   | h = 0 ;                          ▷ To avoid reaching border of LCZ zone
40  | end
41  | else
42  |   | h = |R_in^j'| ;                    ▷ (i, j) is leading to a LCZ OUT node
43  | end
44  | if h > 0 then
45  |   | h' = -1/(2h) + 1
46  | end
47  | else
48  |   | h' = 0
49  | end
50  | H[(i, j)] ← h' ;                              ▷ PUT h to H with key (i, j)
51 end
52 return H
```

Choosing the Best Ant

When all ants find an end node $n \in N_e$ and finish their paths, the algorithm selects the best ant $a \in A$, which we call a champion. The process of finding the champion is specified in Algorithm 10. It iterates A and selects the best ant a based on its path $\mathcal{P}[a]$. The best path contains the highest number of yet uncovered LCZ border nodes that are stored in $\mathcal{N}_{in}$ and $\mathcal{N}_{out}$ and has the shortest length–equal to $|\mathcal{P}[a]|$.

The path P of the best ant, together with the LCZ IN nodes it covers in set B_{in}, and LCZ OUT nodes in the mapping $\mathcal{B}_{out}$ is propagated to the main ANT procedure that adds P to the set of test cases T and according to B_{in} and $\mathcal{B}_{out}$ updates the content of mapping $\mathcal{U}$ with yet uncovered LCZ border nodes. When $\mathcal{U}$ is empty, ANT returns the final set of test cases T.

4.4.3 Adapted Genetic Algorithm

Another alternative, based on a different principle, we propose to generate T from G is the Adapted Genetic Algorithm (AGA). It generates T from G, a specified *threshold* and the test coverage criterion $\mathcal{C}$. The implementation of the AGA consists of the elements and actions detailed in the below paragraph.

Algorithm 9: $ManageBorderNodes(n_{in}, n_{out}, B_{in}, \mathcal{B}_{out}, \mathcal{U}, \mathcal{C}, P_L)$: Cover the LCZ IN and OUT nodes in the parameters by adding them to B_{in} and $\mathcal{B}_{out}$.

Input : LCZ IN node n_{in} to be covered, LCZ OUT node n_{out} to be covered, a set of covered LCZ IN nodes B_{in}, a mapping of covered LCZ OUT nodes $\mathcal{B}_{out}$, a mapping $\mathcal{U}$ of uncovered LCZ border nodes, coverage criterion $\mathcal{C}$, a set P_L where a path through LCZ L is stored

Output: Updated sets B_{in} and $\mathcal{B}_{out}$

if $n_{in} \in \mathcal{U}$ **then**
  if $n_{out} \in \mathcal{U}[n_{in}]$ **then**
    $\mathcal{B}_{out}[n_{in}] \leftarrow \mathcal{B}_{out}[n_{in}] \cup \{n_{out}\}$
    if $n_{in} = n_{out}$ ***and*** $n_{in} \in \mathcal{U}[n_{in}]$ ***and*** $|P_L| > 0$ **then**
      $\mathcal{B}_{out}[n_{in}] \leftarrow \mathcal{B}_{out}[n_{in}] \cup \{n_{in}\}$; ▷ n_{in} `pointing to itself in` $\mathcal{U}[n_{in}]$
    end
  end
end
if $\mathcal{C} = EachBorderOnce$ ***or*** $\mathcal{C} = ComprehensiveEachBorderOnce$ **then**
  for *each* $u_{in} \in \mathcal{U}$ **do**
    if $n_{out} \in \mathcal{U}[u_{in}]$ **then**
      $\mathcal{B}_{out}[u_{in}] \leftarrow \mathcal{B}_{out}[u_{in}] \cup \{n_{out}\}$
    end
  end
  $B_{in} \leftarrow B_{in} \cup \{n_{in}\}$
end
else
  if $n_{in} \in \mathcal{B}_{out}$ **then**
    if $\mathcal{U}[n_{in}] \subset \mathcal{B}_{out}[n_{in}]$; ▷ `All LCZ OUT nodes in` $\mathcal{U}[n_{in}]$ `were covered`
    then
      $B_{in} \leftarrow B_{in} \cup \{n_{in}\}$
    end
  end
end
return $(B_{in}, \mathcal{B}_{out})$

Initially, the algorithm generates a *population* of structures called *chromosomes*. A chromosome represents a candidate solution to a given problem, the quality of which can be assessed by a specific *fitness function*. During the execution of the AGA, a specific *selection mechanism* is implemented for *reproduction* of the members of the population that have the highest value of the fitness function. Afterward, some *genetic operators* that alter the internal parts of the chromosomes, called *genes*, are applied to create better successors of the current population [68]. The selection, reproduction, and genetic operators' application are repeated a specific number of times, which represents the evolution of chromosomes. When the best chromosome in the resultant population (explained further) represents a valid test case, it is added to T. After a number of repetitions, T contains test cases that satisfy the selected test coverage criterion $\mathcal{C}$. The following sections specify

Algorithm 10: *FindChampion*$(A, \mathcal{N}_{in}, \mathcal{N}_{out}, \mathcal{P})$: Iterate a set of ants A and find the one that visits the biggest number of uncovered yet LCZ border nodes using the smallest number of steps.

Input : Set of ants A, mapping of LCZ IN nodes $\mathcal{N}_{in}$ visited by each ant, mapping of LCZ OUT nodes $\mathcal{N}_{out}$ visited by each ant, mapping of each ant paths $\mathcal{P}$

Output: Ant a that found the most efficient path

```
a ← any ant from A
for each t ∈ A do
    if (N_in[t] + N_out[t]) > (N_in[a] + N_out[a]) then
        a ← t
    end
    else if (N_in[t] + N_out[t]) = (N_in[a] + N_out[a]) then
        if |P[t]| < |P[a]| then
            a ← t
        end
    end
end
return a
```

the algorithm details.

The AGA Elements

Chromosome representation. The AGA is designed to find the set of test paths in an SUT model G that satisfies the selected test coverage criterion $\mathcal{C}$. Since the SUT model is a directed graph and the test paths are paths from n_s to $n_e \in N_e$, we decided to model the individual G edges as genes and the chromosomes as sequences of these edges.

The initial population. The initial population can be created in a number of ways:

1. Create the candidate solutions manually [73].
2. Generate the candidate solutions randomly [75].
3. Generate the initial population using a specific heuristic, for example, using chaotic maps or the Min-Min heuristic [104].
4. Generate only a sub-solution and, through a specific operator, improve the population to become a valid solution to the problem [74].

We generate the initial population using the fourth option from the list and more details we present in Section *The Initial Chromosome Population*.

Fitness function. A fitness function of the GA represents an evaluation mechanism that measures chromosome quality. Therefore, it guides the evolution of the chromosomes,

nourishes chromosomes with better fitness, and ignores those with worse fitness. Selecting the right fitness function is crucial to the quality of the solution and the execution run time [105].

To generate chromosomes, which are adjacent sequence of edges in G that satisfy the selected test coverage criterion, several fitness function calculations can be used. Hoseini and Jalili suggest counting the number of prime paths contained in the chromosome combined with its length to cover the prime path coverage [76]. Girgis defines a calculation of the fitness function for solving the DFT problem as a fraction of the number of *def-use* paths covered by an evaluated chromosome to the total number of *def-use* paths in G [106]. To satisfy the coverage of the basis path, Ghiduk defines the fitness value for each chromosome as the sum of adjacent edges of the chromosome divided by the total number of edges on the same chromosome [74].

To make the test set T generated by the AGA satisfying the selected test coverage criterion $\mathcal{C}$, we define the fitness value as $f(c) = \frac{f_a(c) + f_q(c)}{2}$, where f_a is an *adjacent fitness* and f_q is a *quality fitness*. The adjacent fitness $f_a(c)$ of chromosome c is calculated as a fraction of the longest adjacent sequence of edges in c to the total number of edges in c. The quality fitness $f_q(c)$ of chromosome c is calculated as a fraction of the number of LCZ border nodes contained in c that satisfy the selected test coverage criterion $\mathcal{C}$ to the total number of remaining LCZ border nodes not yet contained in the set of chromosomes generated in previous repetitions of the AGA.

Selection step. The AGA selection step is responsible for selecting which individual chromosomes are suitable for reproduction and for the creation of new chromosomes that are passed on to the next generation.

We chose to use roulette wheel selection, which selects a chromosome, from each generation, with a probability $p(i)$ proportional to the fitness of the individual divided by the sum of the fitness of the entire population [107].

Reproduction (crossover). Generally, the reproduction (or crossover) process describes the creation of new chromosomes, called offspring, from the selected pair of parents and is performed for each chromosome in the generation with probability P_c.

There are several methods to perform reproduction, and the differences lie in which of the parents' genes are passed to the offspring [104]. With AGA, we use the one-point (or single) crossover, where a selected pair of parents exchange information around a random position and thus create two new chromosomes.

Another GA operators. In the literature, another GA operators, such as Mutation, Breeding, or Elitist are defined to alter chromosomes and, therefore, facilitate the creation of better candidate solutions to the given problem [74], [104]. The *Mutation* operator iterates genes on the chromosome and, with a specified probability, alters each gene to

a different value. If the offspring contains any chromosomes with lower fitness than the chromosomes with the highest fitness in the current generation, the *Elitist* operator swaps those chromosomes and propagates the individual with the highest fitness in a generation to the next generation.

During the evolution, among other requirements, we need to find a solution that represents a valid test path. However, in the initial population, the chromosomes consist of only a few edges that, in most cases, are non-adjacent. To address this issue, we propose the use of two operators with a certain level of probability to be applied to the chromosomes. The *Breeding* operator iterates the sequence of edges in the chromosome, and when a pair of neighboring edges is non-adjacent, it inserts a random edge between them, which is adjacent to the first edge in the pair. As we cannot know in advance the exact length of the chromosome, there needs to be a way to cut the exceeding part of the chromosome if it gets too large during the evolution. Therefore, we present a *Dissolve* operator that removes all genes after the first occurrence of a non-adjacent pair of neighboring genes in the chromosome.

The stop conditions. The AGA is successfully terminated when the generated T satisfies the test coverage criterion, or fails when the maximum number of repetitions is reached. Although variants of GAs generating test sets that satisfy the segment, branch, and path test coverage criteria [108] already exist, in this thesis we present the novel AGA to generate T, which satisfies the test coverage criteria defined in Section 4.2.

The AGA Variables

The following constants and variables are used in the AGA:

- $REPS$: The maximal number of repetitions of the AGA when it does not return T, which satisfies the selected coverage criterion.
- $ITERS$: The maximal number of generations reached.
- $CMULT$: The multiplication constant that affects the number of chromosomes in a generation.
- $CMIN$: The minimal number of chromosomes in a generation.
- $CMAX$: The maximal number of chromosomes in a generation.
- P_b: The probability of breeding a chromosome in a generation.
- P_d: The probability of dissolving a chromosome in a generation.
- P_c: The probability of crossover of chromosome in a generation.

Constants	Initial value
$REPS$	1000
$ITERS$	500
$CMULT$	10
$CMIN$	20
$CMAX$	100
P_b	0.8
P_d	0.6
P_c	0.6
P_m	0.6
P_g	0.05

Table 4.3: The initial values of the variables and the values of the constants for the AGA algorithm.

- P_m: The probability of applying the mutation operator to a chromosome.
- P_g: The probability of gene mutation in a chromosome.
- f_a: The adjacent fitness value of a chromosome.
- f_q: The quality fitness value of a chromosome.

The initial values of particular variables and values of constants used in AGA are specified in Table 4.3. They are the results of repeating the execution of AGA with the different values of these variables using 310 SUT models presented in Section 7.2. Before the start of this tuning, the initial values of discussed variables and constants were inspired by previous research on GAs [68], [109].

The Main Algorithm

We specify $AGAMain$, the main routine of the AGA, in Algorithm 11. The algorithm accepts an SUT model G, selected $threshold$, the coverage criterion $\mathcal{C}$, and a mapping $\mathcal{U}$ of LCZ border nodes that must be covered. $\mathcal{U}$ is a mapping where a key is an LCZ IN node $k \in in(G, threshold)$ and a value is a set of LCZ OUT nodes $O \in out(G, threshold)$ for which a path from k to $o \in O$ in G exists.

In the initial phase of the $AGAMain$, a new generation of a set of chromosomes Γ is initialized, using the $InitNewGeneration$ procedure (specified in Algorithm 12). The evolution of chromosomes is performed through the $GenerateNextGeneration$ procedure (specified in Algorithm 13) that repeatedly generates the next generation of chromosomes until the count of these repetitions reaches the value of the $ITERS$ constant. When this happens, the $EvalGeneration$ procedure is called to find a new test path that covers some of uncovered LCZ border nodes from the last generation. An LCZ border node

is uncovered, when it is present in the mapping $\mathcal{U}$. If the last generation contains a chromosome that satisfies this condition, the set of test paths T and the mapping of uncovered LCZ border nodes $\mathcal{U}$ are updated, the chromosome generation Γ is reverted to its initial state, and the outer loop repeats. If the last generation does not contain a chromosome that covers any of LCZ border nodes, the algorithm continues the evolution another $ITERS$ times. If this does not result in a solution after $REPS$ repetitions, the $AGAMain$ ends with a failure.

The Algorithm 11 successfully terminates and returns the set of test paths T that satisfies $\mathcal{C}$, when the mapping $\mathcal{U}$ is empty.

Algorithm 11: $AGAMain(G, threshold, \mathcal{C}, \mathcal{U})$: The main AGA algorithm routine.

Input : SUT model G, *threshold*, coverage criterion $\mathcal{C}$, and mapping of uncovered LCZ border nodes $\mathcal{U}$
Output: set of test paths T

```
1  reps = 0 ;                                  ▷ Counter of the algorithm repetitions
2  Γ ← InitNewGeneration(G) ;                  ▷ An initial chromosomes generation Γ
3  T ← an empty set of test paths
4  while reps < REPS ∪ U ≠ ∅ do
5      genNo = 0
6      while genNo < ITERS ∪ U ≠ ∅ do
7          genNo = genNo + 1
8          if genNo = ITERS then
9              (T',U') ← EvalGeneration(G, threshold, C, Γ, U, T)
10             if |T'| > |T| then
11                 T ← T' ; U ← U'
12                 if U ≠ ∅ then
13                     genNo = 0
14                     Γ ← InitNewGeneration(G)
15                 end
16             end
17         end
18         else
19             if |Γ| ≠ Original size of Γ then
20                 Γ ← Γ appended with chromosomes that were added to the initial
                     generation to reach original size of Γ
21             end
22             Γ ← GenerateNextGeneration(G, threshold, C, U, Γ)
23         end
24     end
25 end
26 return T
```

The Initial Chromosome Population

We describe the *InitNewGeneration* procedure that creates an initial generation of chromosome population Γ in Algorithm 12. Initially, it calculates the value of variable *count* that will represent the number of chromosomes in Γ. The value of *count* variable is set according to the number of edges outgoing n_s stored in E_s, the number of all edges incoming to all $n_e \in N_e$ stored in E_e, and the $CMULT$ constant. However, the values of *count* are bound to the $< CMIN, CMAX >$ interval, therefore, if the result is lower, *count* is set to $CMIN$ and if the result is greater, *count* is set to $CMAX$.

Then, the *InitNewGeneration* procedure performs a chromosome creation *count* number of times. During this phase, firstly, the algorithm creates a sequence of edges P that consists of one of the edges outgoing from n_s and one of the edges incoming to one of the $n_e \in N_e$ and this set of edges is set as the initial chromosome c. Secondly, using the *Breeding* procedure specified in Algorithm 15, we apply the Breeding operator and create chromosome c'. Lastly, c' is added to Γ.

Chromosome Evolution

The chromosomes evolve by repetitively creating an offspring generation out of the current one, which is performed in the *GenerateNextGeneration* procedure, described in Algorithm 13. At first, it calls the *Select&Reproduce* procedure (see Algorithm 14) that selects the best chromosomes in Γ and reproduces them to the newly created offspring generation Δ. Then, it calls the *Elitist* function (not detailed by a pseudocode in this thesis), which replaces the worst chromosome in the offspring generation with the best chromosome in the current generation, if it has a lower fitness value. The resulting offspring generation is stored in the new variable Δ'.

Then, the *GenerateNextGeneration* procedure calculates a tuple of fitness values (f_a, f_q) for every chromosome c in Δ'. If c is a test path, but does not cover any uncovered LCZ border nodes stored in the mapping $\mathcal{U}$, c is removed from Δ'. Otherwise, a pseudo-random number $r_b \in \langle 0; 1)$ is generated, and, if smaller than P_b, the *Breeding* operator is applied. The same process follows for the *Dissolve* operator. Lastly, the *FillInGeneration* procedure is called, which appends new chromosomes to Δ' so that the number of chromosomes in a generation remains the same throughout the AGA's execution.

The *FillInGeneration* procedure (not detailed by a pseudocode in this thesis) creates new chromosomes in a way that they are made of one of the edges outgoing n_s, and the *Breeding* operator is applied to them once.

Algorithm 12: *InitNewGeneration*(G): Creates the initial generation of chromosomes.

Input : SUT model G
Output: The initial generation of chromosomes Γ

$\Gamma \leftarrow \emptyset$; ▷ `Initialize an empty generation of chromosomes`
$E_s \leftarrow$ Get edges outgoing from node n_s
$E_e \leftarrow \emptyset$; ▷ `Initialize an empty set of edges`
for $j \in N_e$ **do**
  $E_e \leftarrow E_e \cup$ All edges incoming to the end node j
end
$count = |E_s| \cdot |E_e| \cdot CMULT$
if $count < CMIN$ **then**
  $count = CMIN$
end
else if $count > CMAX$ **then**
  $count = CMAX$
end
else if $count \mod 2 \neq 0$; ▷ `Make` *count* `even, if it is odd`
then
  $count = count + 1$
end
while $i < count$ **do**
  for $(i, j) \in E_s$ **do**
    for $(k, l) \in E_e$ **do**
      $P \leftarrow \emptyset$; ▷ `An empty sequence that will contain edges`
      Add (i, j) to P ; Add (k, l) to P
      $c \leftarrow$ Create chromosome with initial sequence of edges P
      $c' \leftarrow Breeding(c)$
      Put c' to Γ
      $i = i + 1$
      if $i = count$ **then**
        return Γ
      end
    end
  end
end
return Γ

Algorithm 13: $GenerateNextGeneration(G, threshold, \mathcal{C}, \mathcal{U}, \Gamma)$: Generate the next generation of chromosomes.

Input : SUT model G, threshold, coverage criterion $\mathcal{C}$, a mapping of uncovered LCZ border nodes $\mathcal{U}$, and chromosomes in generation Γ
Output: Next generation of chromosomes Δ

$\Delta \leftarrow Select\&Reproduce(G, threshold, \Gamma, \mathcal{C}, \mathcal{U})$
$\Delta' \leftarrow Elitist(\Delta)$
for $c \in \Delta'$ **do**
  $(f_a, f_q) \leftarrow CalculateFitness(G, threshold, c, \mathcal{C}, \mathcal{U})$
  if $f_a = 1 \wedge f_q = 0$; ▷ `Sequence of edges of` c `is adjacent, but doesn't cover any LCZ border nodes`
  then
    Remove c from Δ'
  end
  else
    r_b = A random number from $\langle 0;1 \rangle$
    if $r_b < P_b$ **then**
      $c' \leftarrow Breeding(c)$
      Replace chromosome c by c' in generation Γ
    end
    r_d = A random number from $\langle 0;1 \rangle$
    if $r_d < P_d$ **then**
      $c' \leftarrow Dissolve(c)$
      Replace chromosome c by c' in generation Γ
    end
  end
end
$\Delta \leftarrow FillInGeneration(\Delta')$
return Δ

Reproduction of the Chromosomes

The *Select&Reproduce* procedure, described in Algorithm 14, starts with the Selection and Crossover operators implemented in the *Selection* and *Crossover* procedures.

The *Selection* procedure only sorts chromosomes in Γ according to their fitness, and then returns them as a new generation F.

The *Crossover* procedure creates an offspring generation Δ. It creates a new chromosome cd for each chromosome $c \in F$ with a probability P_c that is made by a one-point crossover of chromosomes c and d; d follows c in F. Otherwise, cd is equal to c. Chromosome cd is then inserted into Δ. *Selection* and *Crossover* procedures are not detailed by a pseudocode in this thesis.

The *Select&Reproduce* procedure continues by calculating the fitness of chromosomes in Δ and storing it into a mapping $\mathcal{F}, \mathcal{F} : c \to x; c \in \Gamma, x \in (0, 1)$. Then, Δ is sorted in descending order (according to the fitness of its chromosomes) and stored into Δ'.

Lastly, with a probability P_m, the Mutation operator, defined in the *Mutation* procedure, is applied to each chromosome $c \in \Delta'$ and the result is put into a new generation Δ''. The *Mutation* procedure (not detailed by a pseudocode in this thesis) iterates each gene g of chromosome c and with a probability P_g swaps g with a different random gene h. In our algorithm, h corresponds to a random edge following gene g's predecessor in the sequence of edges corresponding to c.

We do not further describe the concrete details of *Selection*, *Crossover*, and *Mutation* procedures, as they generally correspond to those already published, for example, by Ghiduk [74].

Extension and Dissolving of the Chromosomes

Due to the nature of the problem, it is impossible to predict the exact length of the chromosome during an iteration. Therefore, it is necessary to have the option to extend or shrink it. The Breeding operator and the Dissolve operator, respectively, represent the solution to this need.

We present the details of the Breeding operator in the *Breeding* procedure, specified in Algorithm 15. It breeds the chromosome c as follows: at first, it iterates the edges in P, which contains the current sequence of edges corresponding to c and increases a counter i. When it reaches the first edge that is non-adjacent with a previous sequence of edges in P, or the end of P, the loop ends. Then, if the index i is smaller than the length of P and if P does not end in node $n \in N_e$, it selects a random edge outgoing of the node with index i in P and places it in P at position $i + 1$. This creates an updated chromosome c', which is then returned.

Algorithm 14: *Select&Reproduce*$(G, threshold, \Gamma, \mathcal{C}, \mathcal{U})$: Selection and reproduction of the chromosomes in the current generation.

Input : SUT model G, *threshold*, chromosomes in a generation Γ, the selected coverage $\mathcal{C}$, and the mapping of uncovered LCZ border nodes $\mathcal{U}$
Output: New generation of chromosomes Δ

$F \leftarrow Selection(G, threshold, \Gamma, \mathcal{C})$; ▷ `Perform the selection of parents`
$\Delta \leftarrow Crossover(F)$; ▷ `A new generation of offspring of` F
$\mathcal{F} \leftarrow$ A mapping $\mathcal{F} : c \rightarrow x; c \in \Gamma, x \in (0, 1)$ of fitness of chromosomes
for $c \in \Delta$ **do**
  $(f_a, f_q) \leftarrow CalculateFitness(G, threshold, c, \mathcal{C}, \mathcal{U})$
  $f = \frac{f_a + f_q}{2}$; $\mathcal{F}[c] = f$
end
$\Delta' \leftarrow$ Sort Δ according to $\mathcal{F}$ in the descending order
$\Delta'' \leftarrow \emptyset$
for $c \in \Delta'$ **do**
  $r_m =$ A random number from $\langle 0;1 \rangle$
  if $r_m < P_m$ **then**
    $c' \leftarrow Mutation(c)$
    Put c' to Δ''
  end
end
$\Delta \leftarrow \Delta''$; **return** Δ

The *Dissolve* procedure (Algorithm 16) keeps the sequence of adjacent edges in c and removes the edges that are not adjacent with this sequence. The procedure is analogous to that of the Breeding operator with one difference: instead of extending the chromosome, it removes all edges after the i-th node in P.

The Fitness Calculation

To evaluate the chromosomes and guide the evolution the right way, the *CalculateFitness* procedure (Algorithm 17) that performs the fitness calculation is defined. It returns a tuple (f_a, f_q), where f_a represents an adjacent fitness, and f_q represents a quality fitness that both contribute by one-half to the total fitness f. The adjacent fitness is calculated as the number of the longest sequence of adjacent edges in P, divided by the total length of P. To calculate the quality fitness, the *BorderNodesCover* procedure (Algorithm 19) that returns a tuple $(q, \mathcal{U}')$ is called. In this tuple, variable q represents a number of LCZ border nodes that the current chromosome contains in its P, and $\mathcal{U}'$ is a mapping of uncovered LCZ border nodes if P is used as a test path.

Algorithm 15: *Breeding*(c): Extend the chromosome by one adjacent edge in its first non-adjacent part.

Input : A chromosome c
Output: Extended chromosome c'

$P \leftarrow$ A current sequence of edges in c
$(k, l) \leftarrow$ The first edge in P
$i = 0$
for *each* $(m, n) \in P \setminus \{(k, l)\}$ **do**
  if $l = m$; ▷ The edges are adjacent
  then $i = i + 1$;
  else if $n \in N_e$; ▷ Some end node inside P
  then $i = i + 1$;
  else break ;
end
if $i < |P|$ **then**
  $(m, n) \leftarrow P_i$; ▷ Edge at i-th position of P
  if $n \notin N_e$ **then**
    $(n, o) \leftarrow$ Pick a random edge starting in n
    $P_{i+1} = (n, o)$; ▷ Place (n, o) at $i + 1$ position in P
  end
end
$c' \leftarrow$ A copy of chromosome c
Assign P to chromosome c'
return c'

Processing the Last Generation to Find Test Paths

When the chromosomes evolve enough, it is necessary to find the best chromosome that represents a valid test path and that contains some of the uncovered combinations of LCZ border nodes. We specify this functionality in the *EvalGeneration* (Algorithm 18). At first, it sorts the chromosomes in Γ in descending order according to their fitness. Then, the *EvalGeneration* iterates over them, searching for the best chromosome c_b that has a quality fitness greater than 0 and contains a sequence of adjacent edges that starts at a start node n_s and ends at one of the $n \in N_e$ (adjacent fitness is equal to 1). Subsequently, the *EvalGeneration* invokes the *BorderNodesCover* procedure (Algorithm 19) on chromosome c_b. The *BorderNodesCover* procedure calculates the number of LCZ border nodes that c covers. Then, the set of test paths T is extended by c_b.

Lastly, the *EvalGeneration* procedure returns a tuple $(T', \mathcal{U}')$, where T' represents an updated set of test paths, and $\mathcal{U}'$ represents an updated mapping of uncovered LCZ border nodes.

Algorithm 16: *Dissolve*(c): Remove all edges that are after the adjacent sequence of edges from the chromosome.

Input : A chromosome c
Output: Chromosome c' with edges that are only adjacent

```
P ← A current sequence of edges in c
(k, l) ← The first edge in P, outgoing from k and incoming to l, {k, l} ∈ N
i = 0
for (m, n) ∈ P \ {(k, l)} do
    if l = m ;                                   ▷ The edges are adjacent
     then i = i + 1 ;
    else break ;
end
P' ← A copy of P
for (m, n) ∈ (P_{i+1}, P_{i+2}, ..., P_{|P|}) do
    P' ← P' \ (m, n) ;                           ▷ Remove (m, n) from P'
end
c' ← A copy of chromosome c
Assign P' to chromosome c'
return c'
```

Covering the LCZ Border Nodes

The *BorderNodesCover* procedure, described in Algorithm 19 performs the evaluation of the quality of the chromosome according to the selected test coverage criterion $\mathcal{C}$, and possible removal of the LCZ border nodes being covered according to the mapping of uncovered LCZ border nodes $\mathcal{U}$.

In its first part, it initializes temporary variables $P, B_{in}, \mathcal{B}_{out}$ and $\mathcal{U}'$, where P is a sequence of edges in c, B_{in} is a set of LCZ IN nodes that P covers, $\mathcal{B}_{out}$ is a mapping of LCZ OUT nodes that P covers, and $\mathcal{U}'$ is a copy of the mapping of uncovered LCZ border nodes $\mathcal{U}$.

Then, the *BorderNodesCover* procedure iterates over the edges in the sequence P corresponding to the input chromosome c. When the traversed edge $(k, l) \in P$ is a LCZ edge and begins in an uncovered LCZ IN node $k \in \mathcal{U}'$, the *BorderNodesCover* iterates the subsequence $P' \subset P$ in which the first edge is outgoing from node k.

Until the edges in the subsequence P' are adjacent, the iteration continues by searching for an uncovered LCZ OUT node $n \in \mathcal{U}'[k]$ of an edge $(m, n) \in P'$. Together with an LCZ IN node k, an LCZ OUT node n composes an uncovered LCZ border node combination, which is stored in a mapping $\mathcal{B}_{out}$. Then, if the input flag r is set, the *BorderNodesCover* procedure removes n from $\mathcal{U}'[k]$.

Next, for *EachBorderOnce* or *ComprehensiveEachBorderOnce* test coverage criteria, the procedure also inserts the LCZ OUT node n into the set $\mathcal{B}_{out}[u_{in}]$ for all LCZ IN

Algorithm 17: $CalculateFitness(G, threshold, c, \mathcal{C}, \mathcal{U})$: Calculate fitness values of the chromosome.

Input : SUT model G, $threshold$, chromosome c, the selected coverage $\mathcal{C}$, and mapping of uncovered LCZ border nodes $\mathcal{U}$.

Output: A tuple of values (f_a, f_q), where f_a represents the adjacency fitness and f_q represents the quality fitness.

$P \leftarrow$ a sequence of edges in c
$a =$ the number of adjacent edges in the longest adjacent sequence of edges in P
$f_a = \frac{a}{|P|}$
if $|\mathcal{U}| > 0$ **then**
    $(q, \mathcal{U}') \leftarrow BorderNodesCover(G, threshold, c, \mathcal{C}, \mathcal{U}, false)$
    $f_q = \frac{q}{|\mathcal{U}| \cdot |P|}$
end
else $f_q = 1$;
return (f_a, f_q)

Algorithm 18: $EvalGeneration(G, threshold, \mathcal{C}, \Gamma, \mathcal{U}, T)$: Iterate over the chromosomes and find the best that represents a valid test path and then cover the border nodes.

Input : SUT model G, $threshold$, coverage criterion $\mathcal{C}$, generation of chromosomes Γ, mapping of uncovered LCZ border nodes $\mathcal{U}$ and a current set of test paths T.

Output: A tuple $(T', \mathcal{U}')$ comprised from an extended set of test paths T' and edited mapping of uncovered LCZ border nodes $\mathcal{U}'$

$C' \leftarrow$ Sorted chromosomes from Γ according to the fitness in descending order
$c_b \leftarrow$ Init an empty chromosome
for *each* $c \in C'$ **do**
    $(f_a, f_q) \leftarrow CalculateFitness(G, threshold, c, \mathcal{C}, \mathcal{U})$
    $P \leftarrow$ A current sequence of edges in c
    $p \leftarrow$ the last node in P
    if $f_a = 1 \wedge f_q > 0 \wedge p \in N_e$ **then**
        $c_b \leftarrow c$
        break
    end
end
$(q, \mathcal{U}') \leftarrow BorderNodesCover(G, threshold, c_b, \mathcal{C}, \mathcal{U}, true)$
$P_b \leftarrow$ A sequence of edges in c_b
$T' \leftarrow T$; Put P_b to T'
return $(T', \mathcal{U}')$

nodes $u_{in} \in \mathcal{U}'$, for which holds that $n \in \mathcal{U}'[u_{in}]$. Moreover, if the flag r is set, the $BorderNodesCover$ procedure also removes n for all u_{in} from $\mathcal{U}'[u_{in}]$.

Then, the algorithm verifies if P reaches one of $n_e \in N_e$ (which can be an LCZ OUT node). If P reaches such a node, the LCZ border nodes covered by P are stored in B_{in} and $\mathcal{B}_{out}$ and if the flag r is set, the LCZ border node combinations present in P are removed from $\mathcal{U}'$, according to the selected test coverage criterion the same way as described above.

When the chromosome iteration is over, the algorithm counts the uncovered LCZ IN nodes stored in B_{in} and saves this value in the variable q_{in}. And if the flag r is set, it removes them from $\mathcal{U}'$. Then, the uncovered LCZ OUT nodes stored in $\mathcal{B}_{out}$ are counted and their number is saved to q_{out}. Lastly, the variable q contains the sum of q_{in} and q_{out}, and is returned together with the updated mapping of the uncovered LCZ border nodes $\mathcal{U}'$.

The repeated chromosome evolution is finished when the generated set of test paths T satisfies the test coverage criterion $\mathcal{C}$ (ensured by the condition that the set of uncovered LCZ border nodes $\mathcal{U}$ is empty). When this condition is satisfied, the main routine of AGA (Algorithm 11) successfully terminates and returns T.

4.4.4 Complexity of the Proposed Algorithms

We present worst-case time and space complexities of the proposed algorithms in Table 4.4. In the table, variable E_s represents a set of all edges outgoing n_s and E_e is a set of all edges incoming all N_e for the corresponding problem instance.

Name	Time Complexity	Space Complexity
SPC	$O(\|\mathcal{L}\| \cdot \|in(G)\| \cdot \|out(G)\| \cdot \|N\|^2)$	$O(\|in(G)\| \cdot \|out(G)\| \cdot \|N\|)$
ANT	$O(\|in(G)\| \cdot \|out(G)\| \cdot \|N\| \cdot \|E\|)$	$O(\|N\|^2)$
AGA	$O(\|E_s\| \cdot \|E_e\|(log(\|E_s\| \cdot \|E_e\|)))$	$O(\|E_s\| \cdot \|E_e\| + \|N\| + \|E\|)$

Table 4.4: Time and space complexity of the proposed algorithms.

Algorithm 19: $BorderNodesCover(G, threshold, c, \mathcal{C}, \mathcal{U}, r)$: Identifies the LCZ border nodes present in c according to the test coverage criterion $\mathcal{C}$, calculates their number, and if r is set, removes them from $\mathcal{U}$.

Input : SUT model G, $threshold$, chromosome c, the selected coverage $\mathcal{C}$, mapping of uncovered LCZ border nodes $\mathcal{U}$, and flag r whether to remove nodes from $\mathcal{U}$.

Output: The tuple $(q, \mathcal{U}')$, where q is the number of LCZ border nodes according to the current coverage that chromosome c contains and $\mathcal{U}'$ is the possible edited mapping of uncovered LCZ border nodes.

```
1  P ← A sequence of edges of c
2  B_in ← Initialize an empty set of LCZ IN nodes n ∈ in(G, threshold) ∈ G that are
     covered in P according to a selected C
3  B_out ← Initialize an empty mapping that for LCZ IN node
     n ∈ in(G, threshold) ∈ G returns a set of LCZ OUT nodes
     O ∈ out(G, threshold) ∈ G that are covered in P according to a selected C
4  U' ← A copy of U
5  for (k, l) ∈ P
6      if C = ComprehensiveAllBorderCombinations ∨ C =
         ComprehensiveEachOnce then
7          (i, j) ← preceding_edge_in_P((k, l))
8          if cop((i, j)) >= threshold then continue ;
9      end
10     if cop((k, l)) >= threshold ∧ k ∈ U'
11         P' ← P_k, ..., P_|P| ;  ▷ A sub-sequence of P in which the first edge
             is is outgoing from node k
12         for (m, n) ∈ P'
13             (o, p) ← next_edge_in(P')
14             if n ≠ o ;          ▷ The next edge in P' is non-adjacent to the
                previous one
15             then break ;
16             ... continues on the following page
```

```
13
14
15
16        ... continuation of Algorithm 19
17        if C = AllBorderCombinations ∨ C =
           EachBorderOnce ∨ cop((m, n)) < threshold then
18          Put k to B_in
19          if U'[k] contains n then
20            Put n to B_out[k]
21            if r = true then
22              Remove n from U'[k]
23              if U'[k] = ∅ then Remove k from U' ;
24            end
25            if
               C = EachBorderOnce ∨ C = ComprehensiveEachBorderOnce
               then
26              for each u_in ∈ U' do
27                if U'[u_in] contains n ∨ u_in = n then
28                  Put n to B_out[u_in]
29                  if r = true then
30                    Remove n from U'[u_in]
31                    if U'[u_in] = ∅ ∧ B_in contains u_in then
32                      Remove u_in from U'
33                    end
34                  end
35                end
36              end
37            end
38          end
39          if cop((m, n)) < threshold then break ;
40        end
41      end
42      ... continues on the following page
```

```
40
41  |
42  |   |  ... continuation of Algorithm 19
43  |   |  if j = m ∧ cop((m, n)) >= threshold then
44  |   |  |  if U' contains k then
45  |   |  |  |  Put k to B_in
46  |   |  |  |  if U'[k] contains n then
47  |   |  |  |  |  Put n to B_out[k]
48  |   |  |  |  |  if r = true then
49  |   |  |  |  |  |  Remove n from U'[k]
50  |   |  |  |  |  |  if U'[k] = ∅ then Remove k from U' ;
51  |   |  |  |  |  end
52  |   |  |  |  |  if
    |   |  |  |  |   C = EachBorderOnce ∨ C = ComprehensiveEachBorderOnce
    |   |  |  |  |   then
53  |   |  |  |  |  |  for each n_in ∈ U' do
54  |   |  |  |  |  |  |  if U'[n_in] contains n then
55  |   |  |  |  |  |  |  |  Put n to B_out[n_in]
56  |   |  |  |  |  |  |  |  if r = true then
57  |   |  |  |  |  |  |  |  |  Remove n from U'[n_in]
58  |   |  |  |  |  |  |  |  |  if U'[n_in] = ∅ ∧ B_in contains n_in then
59  |   |  |  |  |  |  |  |  |  |  Remove n_in from U'
60  |   |  |  |  |  |  |  |  |  end
61  |   |  |  |  |  |  |  |  end
62  |   |  |  |  |  |  |  end
63  |   |  |  |  |  |  end
64  |   |  |  |  |  end
65  |   |  |  |  end
66  |   |  |  end
67  |   |  end
68  |   end
69  end
70  q_in = 0
71  for each b ∈ B_in do
72  |   if U' contains b then
73  |   |  if U'[b] = ∅ then
74  |   |  |  q_in = q_in + 1
75  |   |  |  if r = true then Remove b from U' ;
76  |   |  end
77  |   end
78  end
79  q_out = 0
80  for each k ∈ B_out do q_out = q_out + B_out[k] ;
81  q = q_in + q_out
82  return (q, U')
```

Chapter 5

Baseline Algorithms

In the present experiments, we used two types of baselines to compare the SPC, ANT, and AGA algorithms with. The initial baseline in Section 5.1 gives an account of how effective it might be to test the problem presented in this thesis by test paths satisfying the established test coverage criteria, namely, *Edge*, *Edge-pair*, and *TDL 3*.

The second baseline, the Test Requirements-based algorithm (TR) algorithm, utilized an existing algorithm generating test paths for an input set of test requirements. This baseline served as a comparison of the proposed SPC, ANT, and AGA algorithms with an alternative based on the established test requirements approach. This approach is detailed in Section 5.2.

5.1 Initial Baseline

In the initial baseline, we used T satisfying *Edge*, *Edge-pair*, and *TDL 3* coverage criteria, which were selected based on their wide application in system testing projects [12], [31], [34].

To satisfy the *Edge* coverage, each edge of G must be present at least once in at least one $t \in T$ [12]. To satisfy *Edge-pair* coverage, each possible combination of the two adjacent edges in G must be present at least once in at least one $t \in T$ [12].

To satisfy the *TDL 3* coverage, each possible sequence of the three adjacent edges in G must be present at least once in at least one $t \in T$ [34]. *Edge* coverage is equivalent to *TDL 1* and the *Edge-pair* coverage is equivalent to *TDL 2*.

To generate T that satisfies the *Edge*, *Edge-pair*, and *TDL 3* coverage, we used the Process Cycle Test (PCT) algorithm in the Oxygen platform [110].

To evaluate the experiments, we used the test case evaluation criteria $\mathcal{E}$ defined in Table 4.1. Furthermore, a vital indicator in this evaluation is $E(T)$ for evaluating the *EachBorderOnce* criterion and $A(T)$ for evaluating *AllBorderCombinations*, defined as

follows:

$E(T) = \dfrac{b\_nodes(T)}{|in(G, threshold)| + |out(G, threshold)|} \cdot 100\%$ for given $threshold$.

$A(T) = \dfrac{b\_node\_pairs(T)}{b\_node\_pairs(G, threshold)} \cdot 100\%$, where $b\_node\_pairs(T)$ denotes the number of pair combinations of a LCZ IN node with a LCZ OUT node that are present in $t \in T$, and $b\_node\_pairs(G, threshold)$ denotes the number of all possible LCZ IN and LCZ OUT node combinations required to be present in the test paths of T according to the *AllBorderCombinations* criterion for a given $threshold$.

Herein, we considered the $E(T)$ and $A(T)$ to measure the extent to which T satisfying the *Edge*, *Edge-pair*, or *TDL 3* criteria also satisfies the *EachBorderOnce*, *AllBorderCombinations*, *ComprehensiveEachBorderOnce*, and *ComprehensiveAllBorderCombinations* criteria.

Based on the definitions of the *EachBorderOnce* and *AllBorderCombinations* criteria, if T satisfies *EachBorderOnce*, $E(T) = 100\%$. If T satisfies *AllBorderCombinations*, then $E(T) = 100\%$ and $A(T) = 100\%$. Furthermore, if T satisfies *Edge* , *Edge-pair* and *TDL 3*, $E(T) = 100\%$.

The rules applicable on $E(T)$ and $A(T)$ for *ComprehensiveEachBorderOnce* are identical to those for *EachBorderOnce*, and similarly, the rules for *ComprehensiveAllBorderCombinations* are identical to those for the *AllBorderCombinations* criterion.

5.2 Test Requirements-based Algorithm

The TR employs the test requirements concept for T generation. As introduced in Section 2.2, each test requirement $r \in R$ contains a path p that must be present in the final set of the test cases T. In this case, p represents the path from an LCZ IN to an OUT node. First, the TR algorithm creates a set of test requirements R that ensures T satisfies the given test coverage criterion $\mathcal{C}$.

Second, TR uses an existing greedy set-covering algorithm, published by Nan Li *et al.* [13] to solve the minimum cost test paths problem. The core set-covering sub-problem in their algorithm is solved by adapting an approximation algorithm for the shortest superstring problem.The input to the greedy set-covering algorithm is a set of test requirements R and a small set of test paths TP. The *test path* here is a path in G.

5.2.1 The Main Algorithm

The main routine of TR is described in Algorithm 20. It starts with our *GetTestRequirements* procedure to construct a set of test requirements R. Along with an SUT model G, R composes the inputs to Li's *SetCoveringAlgorithm* procedure [13], which combines

each path in R into a single long path called the super-test requirement $\prod$. Subsequently, the *GetSmallSetOfTestPaths* procedure traverses G and generates a set of all possible test paths TP. Ultimately, TP, $\prod$, and G are inputs to Li's *SplitSuperTestRequirement* procedure [13] that, based on TP, splits the super-test requirement $\prod$ into the final set of the test paths T, each of which is adjacent, starts in n_s, and ends in one of $n_e \in N_e$. As the selected test coverage criterion $\mathcal{C}$ is reflected in the set of test requirements R, it is also satisfied by T.

Algorithm 20: TR(G, $\mathcal{C}$, *threshold*): The main routine of the TR algorithm which creates a set of test requirements R to tour through LCZs and using this set, it constructs T as a set of test paths containing these test requirements.

Input : SUT model G, coverage criterion $\mathcal{C}$, *threshold*
Output: set of test cases T

```
1 R ← GetTestRequirements(G, C, threshold)
2 ∏ ← SetCoveringAlgorithm(G, R) ▷ Defined in [13]
3 TP ← GetSmallSetOfTestPaths(G) ▷ Defined in [13]
4 T ← SplitSuperTestRequirement(G, ∏, TP) ▷ Defined in [13]
5 return T
```

5.2.2 Extraction of Test Requirements

Our procedure *GetTestRequirements*, defined in Algorithm 21, iterates $\mathcal{L}$ and determines all the shortest paths from each $x \in in(L)$ to all $y \in out(L)$ for each $L \in \mathcal{L}$.

After selecting the *AllBorderCombinations* coverage criterion, this is the only necessary step. Accordingly, the algorithm adds all the found shortest paths to R; otherwise, if *EachBorderOnce* coverage is selected, the algorithm reduces the set of found paths. First, the algorithm sorts the paths based on their lengths and stores them in a new list $\mathcal{X}$. Second, the algorithm traverses $\mathcal{X}$ and adds to R only those paths that contain only LCZ IN or LCZ OUT nodes and were not added to R yet.

Note that TR can be used as an objective baseline only for the *EachBorderOnce* and *AllBorderCombinations* test coverage criteria. For *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations*, this baseline is not applicable for comparison because these two test coverage criteria cannot be satisfied using the test requirements approach employed in TR (refer to Section 4.2). In principle, we cannot instruct the TR algorithm by which edge the test path shall enter and leave particular LCZ border nodes (which is a part of the definition of the *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations* criteria).

Algorithm 21: *GetTestRequirements*$(G, \mathcal{C},\ threshold)$: Construct a set of test requirements R that would used further in T generation to tour all $L \in \mathcal{L}(G, threshold)$ in the manner that $\mathcal{C}$ would be satisfied.

Input : SUT model G, coverage criterion $\mathcal{C}$, *threshold*
Output: set of test requirements R

$R \leftarrow \emptyset$
if $\mathcal{C} = AllBorderCombinations$ **then**
  for *each* $L \in \mathcal{L}(G, threshold)$ **do**
    for *each* $x \in in(L)$ **do**
      for *each* $y \in out(L)$ **do**
        $R \leftarrow R \cup \{$ the shortest path from x to y leading through nodes inside L $\}$
      end
    end
  end
  return R
end
if $\mathcal{C} = EachBorderOnce$ **then**
  for *each* $L \in \mathcal{L}(G, threshold)$ **do**
    for *each* $x \in in(L)$ **do**
      for *each* $y \in out(L)$ **do**
        $R_l \leftarrow R_l \cup \{$ the shortest path from x to y leading through nodes inside L $\}$
      end
    end
    $\mathcal{X} \leftarrow$ list of paths from R_l sorted by their length in ascending order
    $L_{in} \leftarrow in(L)$, $L_{out} \leftarrow out(L)$
    for *each* $p \in \mathcal{X}$ *starting with the shortest path* **do**
      $p_{in} \leftarrow$ the first node in p
      $p_{out} \leftarrow$ the last node in p
      if $p_{in} \in L_{in}$ **then**
        $L_{in} \leftarrow L_{in} \backslash \{p_{in}\}$, $L_{out} \leftarrow L_{out} \backslash \{p_{out}\}$
        $R \leftarrow R \cup \{p\}$
      end
      else if $p_{out} \in L_{out}$ **then**
        $L_{out} \leftarrow L_{out} \backslash \{p_{out}\}$
        $R \leftarrow R \cup \{p\}$
      end
    end
  end
  return R
end

Chapter 6

Portfolio Strategy

The LNCT considers four distinct test coverage criteria, $\mathcal{C}$. Owing to the problem complexity, variability in the G topology, and the diverse nature of the test set evaluation criteria $\mathcal{E}$, the development of a universal algorithm to compute the best T with the given inputs is difficult for all possible SUT models. Therefore, we propose a *portfolio strategy* for generating the best T for a general G.

The portfolio strategy is defined in Algorithm 22. As inputs, it accepts G, $threshold$, $\mathcal{C}$, and $\mathcal{E}$ and outputs T. In the first segment of its execution, it computes the individual test sets T satisfying the selected coverage criterion $\mathcal{C}$ for the SUT instance G and $threshold$ by all the proposed algorithms: SPC, ANT, AGA, and TR (the last algorithm is executed only if $\mathcal{C}$ is *AllBorderCombinations* or *EachBorderOnce*). Subsequently, it determines the best T considering the test set optimality criterion $\mathcal{E}$.

In this thesis, the run time of the portfolio strategy was calculated as the sum of the run times of all the proposed algorithms, neglecting the additional time required for selecting the best T using $\mathcal{E}$. To optimize the computation run time, the SPC, ANT, AGA, and TR baseline algorithms can be concurrently executed to obtain a portfolio strategy run time that is equal to the sum of duration of the longest-running algorithm for the computed problem and the time required for selecting the best T.

Another possibility for reducing the run time of the portfolio strategy is to selectively execute the algorithms that are most likely to return the best results for the given combination of the selected G properties ($|N|$, $|E|$, $\overline{deg(n)}$, $|N_e|$, $cycles$, $|\mathcal{L}|$, $|in(G)|$, and $|out(G)|$), $\mathcal{C}$, and $\mathcal{E}$. To perform such a selection, we would need to analyze the possible correlation of the properties of T generated by the proposed algorithms for various combinations of input values with the properties of G.

However, compared with the portfolio strategy version presented in Algorithm 22, such a solution would not assure the selection of the best test set T for a general G, and thus, we do not present such an adjustment in this thesis.

Algorithm 22: $Portfolio(G, threshold, \mathcal{C}, \mathcal{E})$**:** Compute T for G and $threshold$ by SPC, ANT, AGA, and TR algorithms and determine the best T by $\mathcal{E}$.

Input : SUT model G, $threshold$, coverage criterion $\mathcal{C}$, and test set optimality criterion $\mathcal{E}$

Output: set of test cases T

$T_{SPC} \leftarrow \emptyset$, $T_{ANT} \leftarrow \emptyset$, $T_{AGA} \leftarrow \emptyset$, $T_{TR} \leftarrow \emptyset$
$T_{SPC} \leftarrow SPC(G, threshold, \mathcal{C})$
$T_{ANT} \leftarrow ANT(G, threshold, \mathcal{C})$
$T_{AGA} \leftarrow AGA(G, threshold, \mathcal{C})$
if $\mathcal{C} = AllBorderCombinations \vee \mathcal{C} = EachBorderOnce$ **then**
    $T_{TR} \leftarrow TR(G, threshold, \mathcal{C})$
    $T \leftarrow$ a test set from $\{T_{SPC}, T_{ANT}, T_{AGA}, T_{TR}\}$ having the best value of given $\mathcal{E}$
end
else
    $T \leftarrow$ a test set from $\{T_{SPC}, T_{ANT}, T_{AGA}\}$ having the best value of given $\mathcal{E}$
end
return T

Chapter 7

Methods of the Experiments

In the experiments, we compared the test cases created by the proposed SPC, ANT, and AGA algorithms with all baselines for a set of 310 SUT models and all test coverage criteria introduced in Section 4.2. To compare the test cases, we used the evaluation criteria defined in Section 4.3, criteria defined in Section 5.1, the run times of the algorithms, and the potential of the test cases to detect artificial defects, further defined in Section 7.3. In this section, we present the experimental method and setup.

7.1 Implementation of Algorithms

The SPC, ANT, AGA, and TR algorithms as well as the portfolio strategy were implemented in the Oxygen[1] system. Oxygen is an open-freeware MBT platform developed by our research group, which allows the modeling of an SUT and the generation of test cases using various implemented algorithms [110]. To allow for the creation of a SUT model G, we extended the graphical editor of the application.

An exemplary SUT model constructed using Oxygen is illustrated in Figure 7.1, which depicts the UML Activity Diagram of a Smart Home inspired by the system proposed by Aravindan *et al.* [111]. The central server of a Smart Home communicates over a network comprising three subsystems: first, a database server in which the nodes *B–C–D–E–F–Q* represent a subprocess handled by this subsystem; second, a central IoT server (the subprocess handled by this subsystem is modeled by nodes *H–I–J–K*); and lastly, a Raspberry Pi connected with sensors and actuators (the subprocess provided by this subsystem is modeled by nodes *N–O–P–END T*). In this example, the nodes and edges were designated with letters and numbers for simplicity; however, for industrial use, the names can be set to any string.

[1]Java 1.8 executable JAR file packed into a ZIP archive available at `http://still.felk.cvut.cz/download/oxygen_lnct.zip`

When the central server communicates with the external subsystems, the probability of a network outage is higher than the *threshold* level. Therefore, LCZ zones are formed and visually separated from the remaining part of the graph using light-brown arrows and borders; moreover, the symbols of the LCZ IN and LCZ OUT nodes are filled with a light-brown background. In the left application panel, the T generated by the SPC algorithm was visible and denoted as the Test situations.

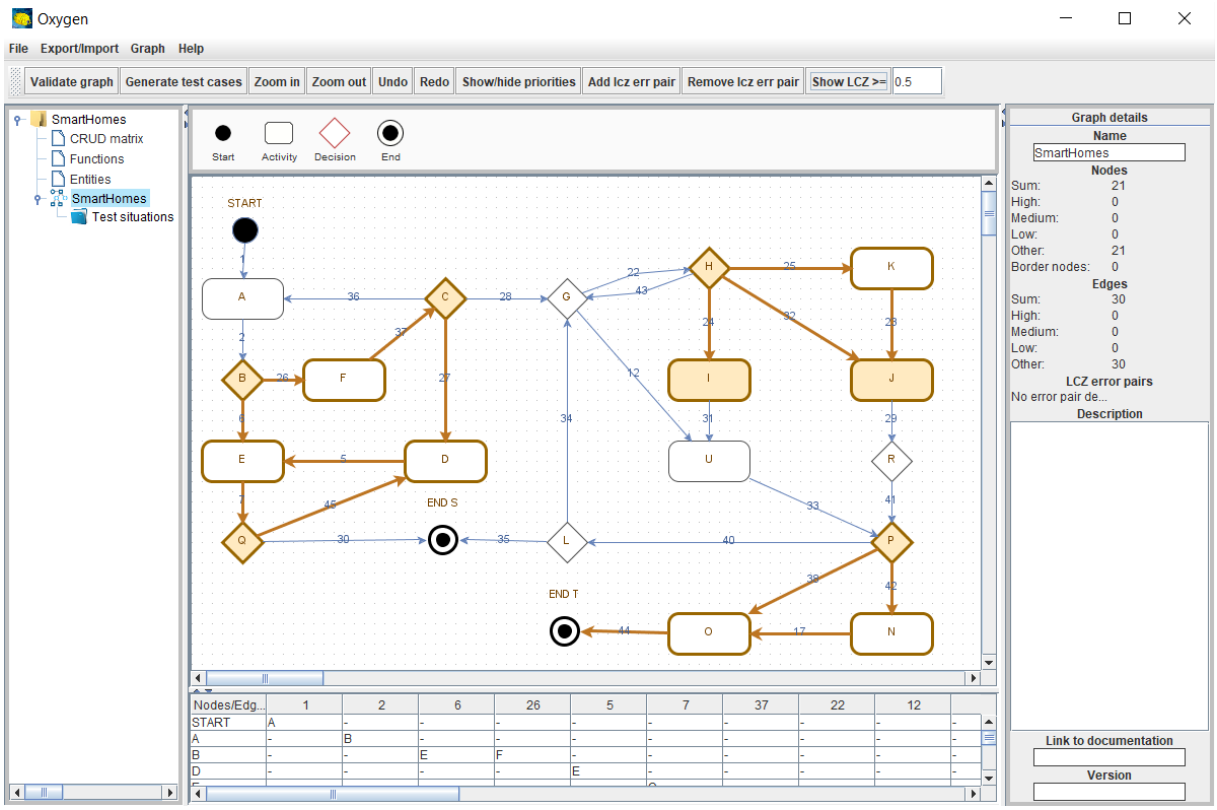


Figure 7.1: Sample SUT model with highlighted LCZs in the Oxygen application.

When the user clicks on the "Test situations" in the left project tree, a pop-up window opens with the individual test cases. If the user selects a set of test cases from the list, the test cases are visually highlighted in the model with bold arrows. In the sample depicted in Figure 7.2, we observe the highlighted test case (composed of nodes *START–A–B–F–C–G–H–J–R–P–O–END T*).

A part of the Oxygen platform development version is a module that compares the algorithms, and we configured this module for LNCT. The comparison module facilitates the execution of multiple algorithms on a given set of SUT models (saved in Oxygen format). After generating the test cases for an SUT model, the module computes the defined properties (here, a set of evaluation criteria; refer to Sections 4.3 and 5.1) and exports the results as a consolidated summary report in a CSV file, which enables further data processing and analysis.

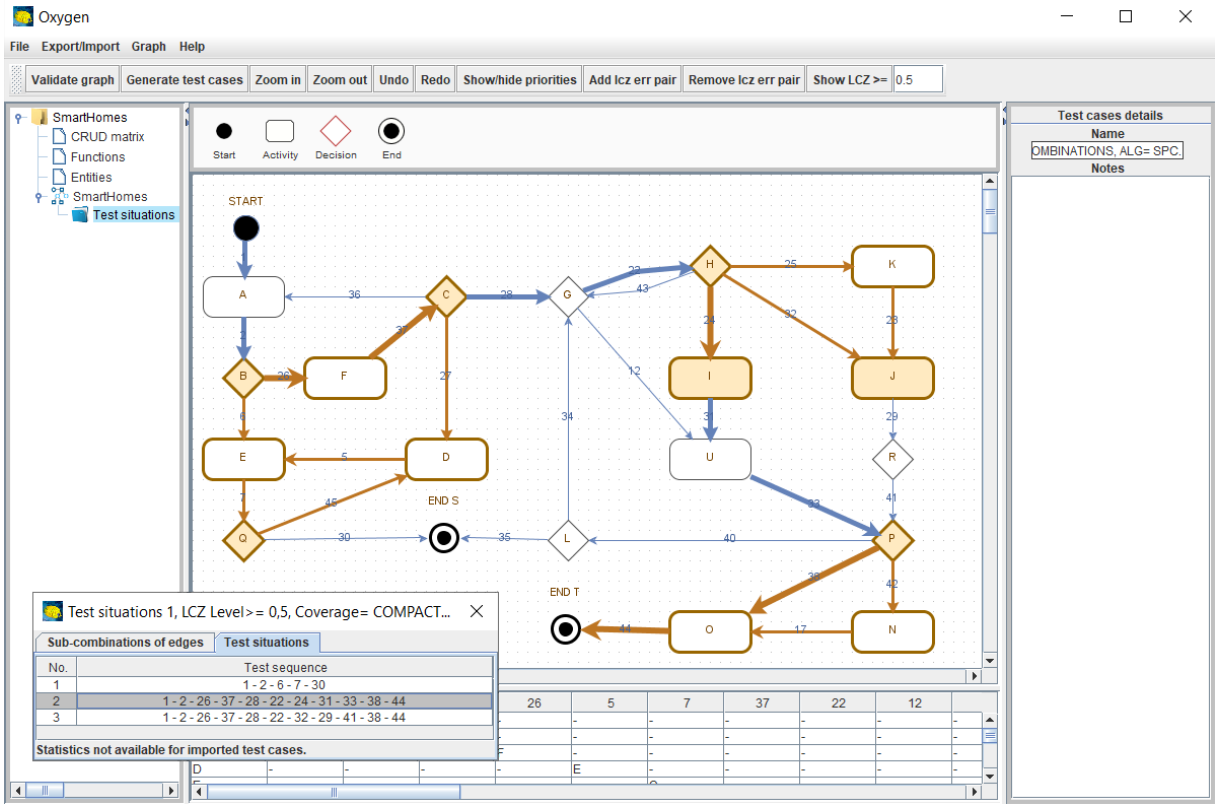


Figure 7.2: Highlighting selected test cases in a SUT model in the Oxygen application.

7.2 Sources of SUT Models used in Experiments

To compare the performance of the algorithms SPC, ANT, and AGA with the baselines, we prepared 310 different SUT models, varying in terms of $|N|$, $|E|$, number of LCZs (denoted as $|\mathcal{L}|$), number of potential LCZ IN and LCZ OUT nodes (denoted as $|in(G)|$ and $|out(G)|$,), number of cycles ($cycles$), average node degree ($\overline{deg(n)}$), and $|N_e|$.

We created 163 models from real IoT projects and added another 147 artificially created or generated models.

Accordingly, the distribution of the nature of sources of the SUT models is visualized in Figure 7.3. The set of SUT models consists of:

1. Three process models and 24 of their modifications from the experimental IoT-based rescue-mission planning and management system for the Czech Police[2] and Mountain Rescue Service, for which our lab created a test strategy and test automation suite;

2. Five process models and 39 of their modifications from our Digital Triage Assistant (DTA)[3] project to implement a sensor network aimed to reduce fatal casualties

[2]https://www.policie.cz/clanek/projekt-patrac.aspx (in Czech)

[3]https://edition.cnn.com/videos/tech/2023/02/20/digital-triage-assistant-soldiers-nato-sensors-orig-contd-zt.cnn

No	Model ID	IoT System Domain	Reference
1	148	Smart Farming	[112]
2	149	Monitoring and Controlling of Sub-station Equipment[5]	[113]
3	153	Electrical Device Surveillance	[114]
4	150	Personal Health Monitoring	[115]
5	151	Smart Parking Reservation System	[116]
6	155	Smart Parking System	[117]
7	154	Patient Health Monitoring System	[118]
8	156	Smart Homes	[111]
9	152	Smart Water Consumption Measurement	[119]
10	157	Smart Washing System of Street Lighting	[120]
11	158	Smart Press Shop Assembly Monitoring	[121]

Table 7.1: Existing third-party IoT systems, whose models were used in experiments.

in defense operations or critical situations in which the army supports the first responders;

3. Three process models and 23 of their modifications from our Teresa[4] project to create a sensor network for the telerehabilitation of post-acute phase COVID-19 patients [16],

4. Eleven models and 55 of their modifications of other real third-party IoT systems described in recently published studies, where a relevant process model was presented (listed in Table 7.1; Model ID refers to Tables 7.3 and 7.4);

5. Artificially created 119 models with a topology resembling those models from previous categories 1.-4., and,

6. Artificially created 28 models by a specialized model generator to achieve variety in the topology of the models used in the experiments.

The model modifications used in set parts 1.-4. were created via modification of the G topology by adding and removing nodes and edges, COP, and relocating the LCZs. As such, the goal was to create a wide variety of problem instances while maintaining a model topology that resembles a real system. In part 4., the LCZs were estimated by analyzing the IoT systems described in studies listed in Table 7.1.

The inputs to the model generator presented in part 6. include $|N|$, $|E|$, $|N_e|$, $cycles$, number of LCZs $|\mathcal{L}|$ and for each LCZ L, the numbers of nodes, edges, cycles, $in(L)$ and $out(L)$. As an output, G is generated.

[4]https://aktualne.cvut.cz/en/reports/20210721-teresa-project-enables-rehabilitation-of-covid-19-patients-in-the-home-environment

[5]power grid elements as transformers, circuit breakers, or relays

Nature of projects 1.-3. render them ideal candidates for LNCT verification. In addition to potentially frequent connectivity outages, Systems 1. and 2. are highly dynamic and have a mission-critical character.

The rescue mission management system supports missions that are generally operated in areas weakly covered by GSM signal, such as forests, mountains, or rural areas. Rescue missions are orchestrated for lost persons who are at risk by staying longer without contact in an uninhabited area (e.g., children, elderly people, or people with a specific medical condition) in situations with no mobile phone to contact them. The system employs GPS trackers for mission participants (humans and search dogs separately) with a dynamic back-end (located in a police or mountain rescue service vehicle). Instead of a GSM network, a mesh network can be used to increase network reliability for communication.

The Digital Triage Assistant represents an even more extreme case in terms of potential network connection unreliability. The current version of the system does not use a GSM network and transmits the data via a proprietary radio channel. The GSM network is allowed only in use cases for an integrated rescue system, where the GSM network is not shut down for security reasons. In addition to network unreliability, a weaker signal can be caused by difficult terrain, military vehicle armoring, or distance (e.g., back-end part of the system mounted in a MEDEVAC helicopter). Moreover, the system components can be damaged or destroyed during a mission or the radio signal can be jammed.

The Teresa project uses a GSM network and a spatially stable back-end. However, various types of outages can hinder the data transmission from patients. For instance, the first controlled study with the patients uncovered certain even comical cases of connectivity outages [16], e.g., a patient lost a smart bracelet while gardening, the wireless connectivity was being disrupted by too many transmitting devices in a makeshift smart home, or several patients were "playing" with the bracelets and mobile phones, which disrupted the configuration of their connectivity to the system. All these situations created great application cases for the LNCT when testing these systems.

7.3 Simulation of System Defects

To evaluate the effectiveness of the generated T for detecting defects caused by limited network connectivity, we extended the SUT model by simulating these defects. Based on the problem described in Section 4, two major situations can occur. First, a defect is present at the border node of the LCZ. This defect is activated when this border node is visited during a process flow in the SUT. Given the test coverage criteria defined in Section 4.2, all these defects will be detected by the LNCT through T satisfying any of the test coverage criteria defined in Section 4.2. Therefore, it is not necessary to include

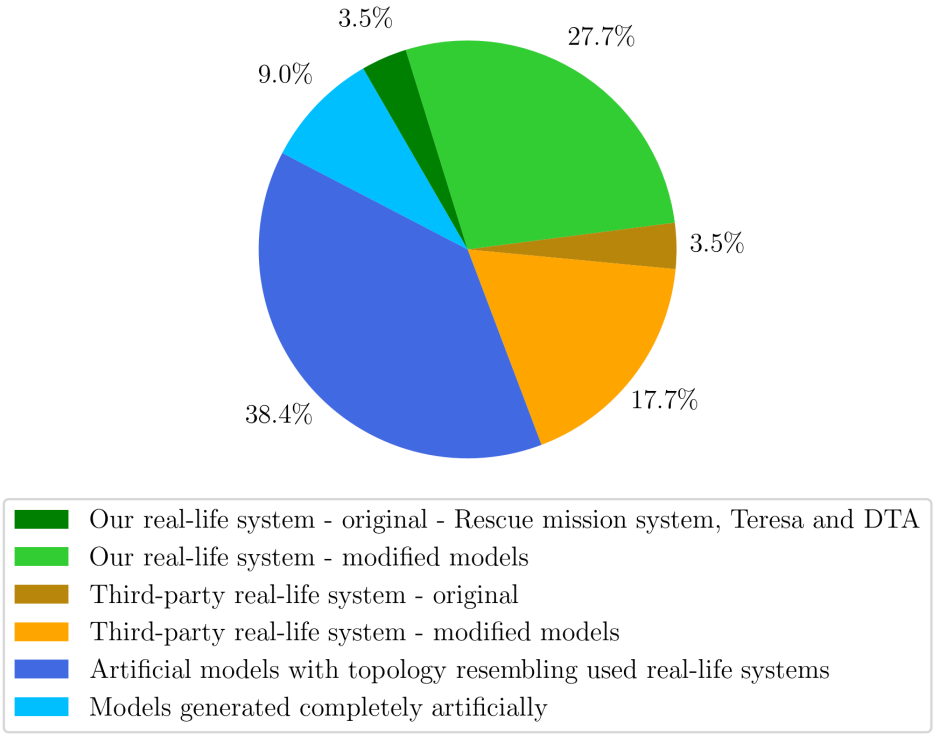


Figure 7.3: Distribution of sources of SUT models used in experiments.

such defects in the evaluation.

Second, a defect can be more complex. For particular *threshold*, it can be simulated as a pair (n_{out}, n_{back}), where $n_{out} \in in(G, threshold)$ represents a node of an SUT process model G in which the network connectivity is disrupted. This impacts the SUT such that when the process flow proceeds to a node $n_{back} \in out(G, threshold)$, the defect is activated and demonstrates itself. We added a set of such simulated defects (n_{out}, n_{back}) to the created SUT models. We denote the set of these defects in an SUT model G as $\chi(G)$.

The number of (n_{out}, n_{back}) defect node pairs in each G was set based on a random number in the interval from $\frac{1}{3}|\mathcal{L}|$ to $\frac{2}{3}|\mathcal{L}|$, and both the interval boundaries were approximated to the nearest integer. In every LCZ with generated defect node pairs, their exact number was equal to a random number in the interval from $\frac{1}{3}$ to $\frac{2}{3}$ of the total number of combinations of LCZ IN and LCZ OUT nodes between whose exists a path. Both interval boundaries were rounded up to the nearest integer.

We inserted artificial defects into SUT models by automated routine modifying the Oxygen project file (based on the XML format) containing the model definitions. A user can review the inserted defects for a particular SUT model in Oxygen application, where these defects can also be defined manually. In the example presented in Figure 7.4, the

green rectangle encompasses an overview of the defect node pairs defined in the SUT model.

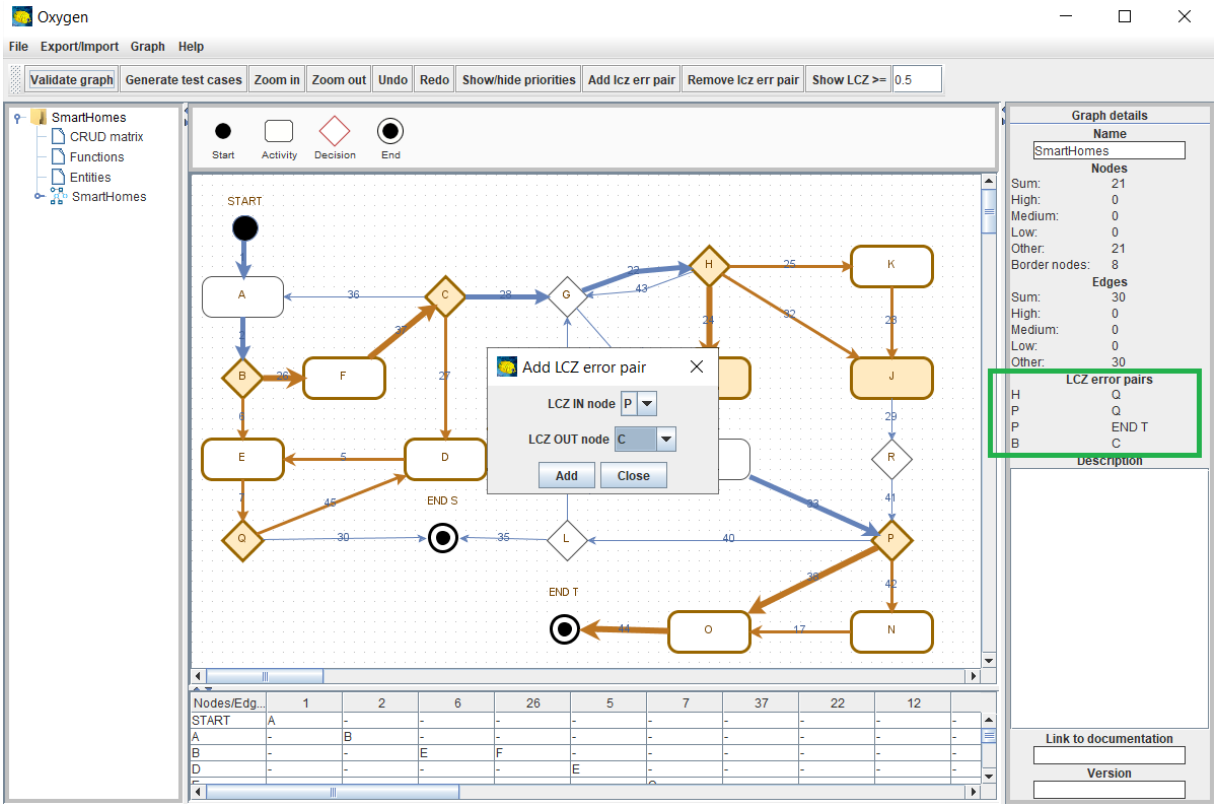


Figure 7.4: Visualization and manual definition of artificial defects in Oxygen application.

To measure the effectiveness of T in detecting these artificial defect node pairs in G, we use $\psi = \frac{activated_T}{|\chi(G)|} \cdot 100\%$, where $activated_T$ denotes the number of artificial defect node pairs activated (visited) by test case $t \in T$. These results must be considered in the context of the size of T, namely, $l(T)$. Here, we considered $\Psi = \frac{\psi}{l(T)}$ as an indicator of the effectiveness of T for activating defect node pairs inserted in G. As such, a higher Ψ indicates better potential for defect detection.

7.4 Detailed SUT Models Properties

The overall properties of the SUT models used in the experiments are summarized in Table 7.2, including the minimal (MIN), maximal (MAX), average ($\overline{x}$), and median ($\widetilde{x}$) values of the individual model properties discussed earlier.

In the experiments, the $cop(e)$ for all LCZ edges was set to 50%, and the *threshold* was set to 50%.

Complete overviews of the SUT models used in the experiments are listed in Tables 7.3 and 7.4.

	$\|N\|$	$\|E\|$	$\overline{deg(n)}$	$\|N_e\|$	$cycles$	$\|\mathcal{L}\|$	$\|in(G)\|$	$\|out(G)\|$	$\|\chi(G)\|$
MIN	11	13	2.11	1	0	1	1	1	1
MAX	325	488	4.26	21	35	6	14	17	23
$\overline{x}$	48.76	73.11	3.00	4.09	5.73	2.37	4.64	6.12	5.85
$\widetilde{x}$	40	61	2.97	4	4	2	4	6	5

Table 7.2: Overall properties of SUT models used in experiments.

7.5 Computation of Test Cases

The computation of T performed on a machine with OS Windows 11, Java version 19.0.1, and the following hardware configuration: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz, 16 GB RAM with an SSD disk. We computed T for all the 310 SUT models introduced in Sections 7.2 and 7.4.

As the SPC, ANT, AGA, and TR algorithms have a nondeterministic nature, the T for these algorithms was computed ten times and the results were averaged.

ID	$\|N\|$	$\|E\|$	$\overline{deg(n)}$	$\|N_e\|$	*cycles*	$\|\mathcal{L}\|$	$\|in(G)\|$	$\|out(G)\|$	$\|\chi(G)\|$	ID	$\|N\|$	$\|E\|$	$\overline{deg(n)}$	$\|N_e\|$	*cycles*	$\|\mathcal{L}\|$	$\|in(G)\|$	$\|out(G)\|$	$\|\chi(G)\|$
1	19	30	3.16	1	5	1	2	1	1	78	25	43	3.44	4	1	2	4	6	7
2	24	38	3.17	2	7	1	2	1	2	79	29	48	3.31	5	0	3	5	8	6
3	24	38	3.17	2	7	1	3	1	2	80	20	40	4	1	10	2	4	2	3
4	30	49	3.27	1	7	2	4	3	3	81	20	40	4	1	12	3	5	4	4
5	30	49	3.27	1	7	1	2	2	1	82	22	37	3.36	3	9	3	7	5	4
6	25	40	3.2	1	7	1	1	3	1	83	30	49	3.27	1	8	3	4	3	4
7	25	40	3.2	1	7	1	4	1	2	84	40	70	3.5	1	15	2	3	2	3
8	33	52	3.15	1	7	2	2	4	3	85	40	81	4.05	2	14	2	4	7	6
9	33	52	3.15	1	7	1	2	1	1	86	24	45	3.75	5	10	2	5	7	8
10	28	46	3.29	4	6	3	5	9	9	87	50	100	4	2	26	3	6	5	2
11	28	48	3.43	4	5	1	1	2	2	88	54	103	3.81	3	21	3	5	6	5
12	25	38	3.04	1	4	1	1	2	1	89	60	104	3.47	1	15	3	5	7	5
13	25	38	3.04	1	4	2	4	2	2	90	33	59	3.58	1	11	1	3	5	8
14	26	41	3.15	1	2	2	4	5	5	91	79	140	3.54	1	25	3	4	3	3
15	25	39	3.12	1	2	2	3	5	5	92	61	105	3.44	1	16	3	6	6	4
16	40	57	2.85	3	3	3	4	7	6	93	42	78	3.71	2	16	1	3	6	7
17	40	57	2.85	3	3	2	6	6	9	94	36	69	3.83	2	18	2	5	6	6
18	40	57	2.85	3	3	2	6	5	9	95	40	69	3.45	1	22	1	1	1	1
19	19	26	2.74	1	0	1	2	1	1	96	50	99	3.96	2	22	3	8	8	7
20	19	26	2.74	1	0	2	2	3	3	97	54	115	4.26	4	22	2	6	14	22
21	30	38	2.53	6	1	1	2	3	2	98	93	170	3.66	2	29	3	8	11	19
22	30	49	3.27	6	4	2	5	7	8	99	30	45	3	2	3	2	4	4	4
23	31	42	2.71	3	1	2	4	2	3	100	24	45	3.75	4	6	2	2	6	5
24	30	47	3.13	7	0	1	1	2	1	101	20	32	3.2	4	6	1	1	2	2
25	30	50	3.33	6	1	1	3	4	5	102	25	46	3.68	3	4	1	3	2	4
26	35	58	3.31	4	3	1	2	2	3	103	25	45	3.6	4	2	2	4	6	8
27	30	48	3.2	3	1	1	4	4	6	104	30	49	3.27	1	0	3	6	6	4
28	30	45	3	1	1	1	2	3	3	105	30	54	3.6	1	4	3	4	3	2
29	30	44	2.93	3	0	1	3	3	5	106	35	51	2.91	4	5	2	3	6	4
30	32	51	3.19	2	2	2	5	5	5	107	35	47	2.69	7	3	1	3	2	2
31	30	43	2.87	2	0	1	2	2	2	108	40	64	3.2	3	0	3	3	9	7
32	30	46	3.07	2	1	2	3	5	3	109	40	60	3	5	3	2	5	2	4
33	34	47	2.76	1	2	1	3	2	3	110	45	72	3.2	4	1	3	3	4	4
34	33	50	3.03	4	2	2	5	3	3	111	45	58	2.58	2	0	1	2	1	2
35	35	58	3.31	1	3	3	6	5	6	112	50	77	3.08	4	4	2	5	5	4
36	31	38	2.45	2	0	1	1	1	1	113	50	90	3.6	4	1	3	6	7	7
37	30	42	2.8	3	1	1	2	2	2	114	45	77	3.42	2	0	3	6	3	4
38	34	55	3.24	4	1	2	6	5	9	115	60	72	2.4	15	2	1	1	3	1
39	31	40	2.58	3	1	1	2	2	1	116	65	117	3.6	3	2	2	3	6	6
40	31	47	3.03	2	0	2	5	6	7	117	70	100	2.86	8	2	2	3	6	5
41	30	46	3.07	3	2	2	3	4	2	118	75	90	2.4	21	3	1	1	1	1
42	35	57	3.26	3	1	2	4	4	4	119	80	97	2.43	15	3	2	2	3	3
43	31	43	2.77	2	1	3	6	5	4	120	47	67	2.85	4	6	3	7	6	10
44	40	54	2.7	3	1	2	5	2	3	121	52	86	3.31	5	11	3	10	11	12
45	40	59	2.95	4	1	1	3	6	6	122	44	63	2.86	4	6	3	10	7	12
46	44	55	2.5	8	1	3	4	4	3	123	37	56	3.03	2	5	2	7	3	4
47	42	56	2.67	4	1	2	5	3	4	124	63	103	3.27	1	5	3	7	8	8
48	40	60	3	1	1	2	2	5	2	125	53	91	3.43	3	2	2	4	4	4
49	44	59	2.68	4	1	3	8	6	6	126	40	62	3.1	3	7	2	5	5	5
50	48	66	2.75	5	2	3	10	5	9	127	93	149	3.2	1	21	3	4	5	3
51	42	59	2.81	4	1	3	6	5	5	128	85	152	3.58	2	21	3	4	5	4
52	40	61	3.05	6	2	3	3	8	4	129	138	234	3.39	2	34	4	7	6	4
53	44	64	2.91	4	2	2	7	7	8	130	50	79	3.16	1	2	3	6	5	5
54	48	81	3.38	4	6	1	6	4	9	131	82	145	3.54	2	6	3	4	7	6
55	40	66	3.3	2	3	2	7	6	8	132	70	93	2.66	3	5	3	4	7	4
56	42	73	3.48	1	3	3	9	9	10	133	75	97	2.59	8	2	3	4	4	4
57	41	64	3.12	1	4	1	4	4	6	134	88	126	2.86	5	4	3	10	6	5
58	45	71	3.16	1	5	2	6	7	9	135	80	122	3.05	6	4	3	3	8	6
59	40	66	3.3	2	1	2	7	5	4	136	85	126	2.96	1	2	2	6	6	9
60	42	64	3.05	2	0	3	5	6	4	137	80	108	2.7	8	4	3	4	5	2
61	41	59	2.88	6	1	3	3	5	4	138	97	144	2.97	3	6	3	6	7	4
62	45	66	2.93	1	1	3	6	6	8	139	92	146	3.17	4	4	4	6	7	5
63	53	75	2.83	3	1	3	8	7	7	140	211	357	3.38	4	25	4	9	7	8
64	52	66	2.54	8	2	3	4	5	3	141	156	242	3.1	1	0	2	6	4	8
65	51	106	4.16	2	1	3	8	13	10	142	271	351	2.59	5	35	4	10	10	12
66	56	91	3.25	3	1	2	6	8	7	143	259	352	2.72	3	16	3	13	13	17
67	54	89	3.3	4	1	3	8	8	8	144	260	320	2.46	1	0	2	4	2	3
68	52	77	2.96	3	1	3	6	7	2	145	325	388	2.39	1	0	2	2	4	3
69	58	92	3.17	2	0	3	4	6	6	146	260	420	3.23	1	0	2	2	4	2
70	50	78	3.12	4	2	3	3	5	4	147	325	488	3	1	0	2	2	2	2
71	35	65	3.71	3	6	2	8	8	14	148	24	28	2.33	3	1	2	3	3	2
72	32	58	3.63	2	5	2	5	3	4	149	20	25	2.5	2	1	2	2	2	2
73	30	58	3.87	2	2	3	7	7	10	150	29	37	2.55	2	0	4	4	4	3
74	33	57	3.45	2	0	3	6	5	4	151	25	28	2.24	2	1	3	3	3	2
75	20	35	3.5	6	5	2	2	4	3	152	26	30	2.31	3	0	5	6	5	4
76	20	32	3.2	5	6	1	1	2	2	153	18	20	2.22	2	0	3	3	3	2
77	25	45	3.6	6	6	1	3	1	2	154	21	24	2.29	4	3	2	2	2	2

Table 7.3: Complete overview of SUT models used in experiments (Part 1).

ID	$\|N\|$	$\|E\|$	$\overline{deg(n)}$	$\|N_e\|$	$cycles$	$\|\mathcal{L}\|$	$\|in(G)\|$	$\|out(G)\|$	$\|\chi(G)\|$	ID	$\|N\|$	$\|E\|$	$\overline{deg(n)}$	$\|N_e\|$	$cycles$	$\|\mathcal{L}\|$	$\|in(G)\|$	$\|out(G)\|$	$\|\chi(G)\|$
155	15	17	2.27	2	0	1	3	3	3	233	57	75	2.63	1	6	3	3	4	3
156	19	21	2.21	2	0	3	3	3	2	234	30	39	2.6	4	5	3	5	7	9
157	18	19	2.11	4	0	1	3	3	2	235	42	54	2.57	6	5	3	5	11	4
158	15	16	2.13	1	0	1	3	2	3	236	49	68	2.78	6	8	3	6	9	2
159	19	24	2.53	4	5	2	2	2	2	237	61	82	2.69	5	7	3	6	7	3
160	24	35	2.92	4	7	3	4	3	2	238	64	104	3.25	8	13	3	8	12	14
161	22	31	2.82	4	6	2	2	2	2	239	12	13	2.17	1	1	1	1	2	1
162	25	46	3.68	3	9	2	3	2	2	240	24	29	2.42	1	2	1	1	2	2
163	19	29	3.05	4	7	1	3	1	2	241	31	37	2.39	1	4	1	1	4	2
164	30	49	3.27	5	10	3	4	7	2	242	72	95	2.64	8	6	3	4	10	4
165	38	61	3.21	6	11	3	4	8	8	243	19	25	2.63	1	3	1	1	2	1
166	42	66	3.14	8	11	2	4	11	8	244	29	41	2.83	2	7	1	1	3	2
167	51	84	3.29	4	12	5	9	13	13	245	38	50	2.63	1	6	1	1	4	2
168	26	37	2.85	4	8	3	4	4	4	246	67	95	2.84	7	7	2	4	9	7
169	34	58	3.41	4	8	3	5	4	3	247	72	92	2.56	12	5	3	4	14	12
170	39	71	3.64	4	11	2	5	4	8	248	16	18	2.25	1	2	2	3	3	2
171	37	63	3.41	3	8	3	5	4	4	249	27	32	2.37	1	2	2	4	3	3
172	42	69	3.29	4	14	3	7	6	10	250	40	52	2.6	3	4	3	5	6	6
173	38	66	3.47	4	10	3	5	7	6	251	61	102	3.34	4	9	3	6	8	11
174	43	75	3.49	4	10	3	7	8	3	252	26	31	2.38	2	3	2	3	4	3
175	46	94	4.09	3	13	3	11	10	12	253	31	41	2.65	2	3	2	4	4	4
176	56	97	3.46	6	11	3	10	14	23	254	45	63	2.8	6	9	3	4	9	7
177	28	37	2.64	5	9	2	4	4	4	255	63	90	2.86	7	8	3	5	11	12
178	35	54	3.09	4	12	1	2	1	1	256	31	38	2.45	5	2	2	3	5	3
179	39	56	2.87	5	10	2	5	5	5	257	40	50	2.5	6	1	2	2	5	4
180	38	58	3.05	4	11	3	5	5	4	258	44	68	3.09	6	3	2	3	7	5
181	50	82	3.28	4	12	2	5	5	6	259	63	82	2.6	8	6	3	4	8	3
182	40	69	3.45	4	11	2	5	8	8	260	71	98	2.76	10	6	4	7	12	11
183	48	80	3.33	8	11	2	5	12	11	261	35	50	2.86	4	2	2	3	4	4
184	53	87	3.28	5	12	2	5	9	9	262	46	67	2.91	5	3	3	6	8	7
185	63	104	3.3	6	13	3	7	12	10	263	53	83	3.13	6	4	3	7	8	3
186	12	14	2.33	1	3	1	1	2	1	264	57	93	3.26	6	11	3	7	6	5
187	21	29	2.76	3	3	1	1	5	2	265	75	114	3.04	7	4	6	13	13	16
188	28	39	2.79	2	6	2	4	4	6	266	41	60	2.93	2	2	5	5	6	6
189	40	62	3.1	8	10	2	2	10	7	267	49	71	2.9	7	1	5	6	9	11
190	51	77	3.02	12	11	3	3	15	6	268	62	104	3.35	6	4	3	14	11	16
191	54	84	3.11	7	10	2	4	10	7	269	69	114	3.3	7	5	3	11	12	9
192	59	87	2.95	9	11	2	3	11	6	270	79	122	3.09	12	4	2	7	12	18
193	81	125	3.09	10	13	4	9	16	9	271	38	49	2.58	2	3	3	6	5	5
194	16	21	2.63	1	4	2	2	4	2	272	51	71	2.78	4	3	2	5	8	13
195	26	41	3.15	3	7	2	2	6	5	273	61	92	3.02	7	6	3	7	12	15
196	39	72	3.69	2	15	3	4	7	6	274	71	102	2.87	10	5	3	8	15	4
197	53	83	3.13	3	7	1	2	6	5	275	83	121	2.92	9	7	3	6	9	12
198	58	94	3.24	6	5	1	4	10	16	276	33	39	2.36	5	0	6	8	7	8
199	65	91	2.8	3	8	4	5	9	7	277	41	56	2.73	4	1	5	7	6	6
200	49	77	3.14	4	12	3	4	8	5	278	50	67	2.68	6	2	4	7	7	6
201	59	108	3.66	4	16	4	9	13	12	279	58	77	2.66	9	3	2	8	14	20
202	61	102	3.34	4	18	4	5	10	7	280	71	98	2.76	9	6	5	8	12	7
203	11	13	2.36	1	2	1	1	1	1	281	31	39	2.52	5	2	3	4	6	5
204	20	27	2.7	1	4	1	1	1	1	282	39	53	2.72	3	1	3	4	7	5
205	31	41	2.65	3	3	2	2	4	3	283	51	69	2.71	6	3	2	5	10	7
206	49	70	2.86	7	8	3	3	8	5	284	62	77	2.48	13	1	3	4	17	7
207	36	48	2.67	5	5	3	5	6	3	285	74	97	2.62	13	4	3	6	15	12
208	47	72	3.06	4	10	2	5	6	6	286	31	47	3.03	5	6	2	4	5	5
209	52	79	3.04	6	5	3	9	10	14	287	39	56	2.87	8	6	2	2	6	2
210	62	99	3.19	3	10	3	9	7	10	288	47	73	3.11	7	10	2	5	9	11
211	71	124	3.49	4	22	3	8	16	20	289	61	87	2.85	10	6	2	5	13	13
212	14	17	2.43	1	3	1	1	1	1	290	81	113	2.79	14	9	3	6	15	9
213	30	41	2.73	3	3	2	3	3	2	291	23	29	2.52	3	1	1	3	6	4
214	44	59	2.68	8	6	2	4	9	11	292	33	46	2.79	5	5	1	5	7	16
215	51	71	2.78	7	8	3	3	8	6	293	42	63	3	5	7	1	7	10	17
216	27	39	2.89	3	6	1	2	4	3	294	51	72	2.82	5	9	2	4	7	7
217	47	66	2.81	4	5	3	5	7	5	295	62	93	3	7	2	2	6	9	11
218	54	80	2.96	8	8	3	7	10	11	296	32	42	2.63	3	2	3	3	5	4
219	59	81	2.75	8	6	4	7	10	10	297	40	56	2.8	4	6	3	5	7	7
220	66	95	2.88	7	12	3	6	12	6	298	45	69	3.07	5	5	3	8	9	10
221	19	25	2.63	1	4	2	2	3	3	299	55	76	2.76	5	3	3	6	10	11
222	29	40	2.76	2	5	3	3	4	3	300	62	85	2.74	7	4	3	7	12	9
223	50	71	2.84	6	8	2	4	7	4	301	26	33	2.54	4	2	1	3	5	3
224	63	92	2.92	5	6	4	4	10	7	302	33	40	2.42	5	2	1	4	5	7
225	34	45	2.65	4	6	2	2	6	4	303	43	57	2.65	10	0	2	5	7	6
226	38	55	2.89	3	8	3	3	5	5	304	50	68	2.72	8	2	2	4	8	6
227	56	77	2.75	5	8	3	7	7	6	305	64	93	2.91	7	3	3	10	7	10
228	71	104	2.93	7	5	4	8	16	18	306	26	32	2.46	2	0	1	3	3	3
229	76	116	3.05	6	13	2	2	8	5	307	32	43	2.69	3	0	2	7	6	10
230	18	22	2.44	1	4	3	3	4	4	308	41	59	2.88	3	2	2	9	8	11
231	34	42	2.47	2	4	3	3	5	3	309	55	71	2.58	5	2	2	8	10	9
232	40	53	2.65	3	4	3	3	6	4	310	62	103	3.32	4	8	3	11	15	8

Table 7.4: Complete overview of SUT models used in experiments (Part 2).

Chapter 8

Results of the Experiments

In this chapter, we describe the results of the SPC, ANT, AGA, and TR algorithms and the results of the initial baseline for the test coverage criteria specified in Section 4.2, and we also present the results of the portfolio strategy.

First, we present the overall results of the algorithms in Section 8.1. Then, Section 8.2 analyzes which algorithms provided the best results for individual SUT models, followed by Section 8.3, which describes effectiveness of detection of simulated limited network connectivity defects in a SUT. In Section 8.4, we present the results of the portfolio strategy. Lastly, Section 8.5 presents the measured data on the time effectiveness of individual algorithms and the portfolio strategy.

8.1 Properties of Test Sets Produced by Compared Algorithms

First, we compare the properties of the individual test sets T generated by the algorithms and baselines using the evaluation criteria $\mathcal{E}$ introduced in Table 4.1. For the *EachBorderOnce* and *AllBorderCombinations* criteria, the results are presented in Table 8.1. For the *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations* criteria, the results are shown in Table 8.2. In Tables 8.1 and 8.2, for each algorithm, the *min*, *avg*, and *max* values of the evaluation criterion $\mathcal{E}$ refer to minimum/average/maximum values of all averaged results of ten repeated executions for all 310 SUT models. Note that values of Ψ presented in Tables 8.1 and 8.2 are not equal to the division of the values of average ψ over average $l(T)$ as presented in these tables, but they are an average of $\frac{\psi}{l(T)}$ for each of the SUT models.

For better visualization of the results, we present them in graphs as well. In Figure 8.1 for *AllBorderCombinations*, in Figure 8.2 for *EachBorderOnce*, in Figure 8.3 for *Compre-*

hensiveAllBorderCombinations, and in Figure 8.4 for the *ComprehensiveEachBorderOnce* criterion. To shorten the captions, in Figures 8.1, 8.2, 8.3, and 8.4, *TDL 1* matches the *Edge* coverage, *TDL 2* matches the *Edge-pair* coverage, and *PS* denotes the portfolio strategy.

For the ***AllBorderCombinations*** test coverage criterion, the ANT algorithm produced T with a visibly lower average $|T|$ than the TR and SPC algorithms; the AGA yielded similar results for this criterion. The ANT algorithm's average $|T|$ was 50% lower than the average $|T|$ of the TR algorithm and 46% lower compared to the SPC algorithm's average, whereas it was only 16% lower than the average $|T|$ of the AGA (see Table 8.1 and Figure 8.1).

The ANT algorithm also outperformed the other algorithms in the average total length of test cases $l(T)$. The ANT algorithm's average $l(T)$ was 31% smaller than the value for the TR algorithm, 30% smaller than for the SPC algorithm, and 10% smaller than for the AGA.

For the $U(T)$ criterion, which represents the ratio of unique edges to the total number of edges in test set T, the ANT algorithm returned the best results. It had the value of average $U(T)$ 6% higher than for the AGA, 29% higher than for the SPC algorithm, and 33% higher than for the TR algorithm.

When looking at the $B(T)$ criterion, which represents an average number of border nodes in T, the ANT algorithm yielded the best result: 9.5% higher than for the AGA, 44% higher than for the SPC algorithm, and 53% higher than for the TR algorithm.

The best results of the evaluation criteria $|T|$, $l(T)$, $U(T)$, and $B(T)$ of T produced by the ANT algorithm were contrasting with the dispersive length of test cases in T produced by this algorithm. When we look at the individual results of the test set length dispersion $s(T)$, where the smallest value means the best result, the average value of results produced by the ANT algorithm was 3.3 times larger than the results by the TR algorithm, which were the best in this category. The AGA produced results with $s(T)$ 69% larger than the results of the TR algorithm, and the SPC algorithm produced results 21% larger than the results of the TR algorithm.

For the ***EachBorderOnce*** test coverage criterion, the results for $|T|$ were similar to those for the *AllBorderCombinations*. The ANT algorithm produced test sets that had the smallest average size: 21% smaller than the average $|T|$ for test sets produced by the AGA, 39% smaller than the average $|T|$ for the SPC algorithm, and more than 47% smaller than the average $|T|$ for the TR algorithm, which, on average, produced test sets with the largest size (see Table 8.1 and Figure 8.2).

When looking at the average $l(T)$, the outcome for the *EachBorderOnce* coverage was similar to the outcome for the *AllBorderCombinations* coverage. For the AGA, the

Table 8.1: Average values of test set evaluation criteria over all G for the *AllBorderCombinations* and *EachBorderOnce* test coverage criteria.

Criterion / alg		$\|T\|$	$l(T)$	$s(T)$	$U(T)$	$B(T)$	$A(T)$	$E(T)$	$\psi(T)$	Ψ	$t[s]$
both **AllBorderCombinations** and **EachBorderOnce**											
Edge		17.25	328.62	8.59	33%	5%	92.3%	100%	95.4%	0.68%	0.016
Edge-pair		27.19	539.27	10.33	20%	3%	96.3%	100%	97.6%	0.43%	0.041
TDL 3		42.44	948.07	11.3	14%	2%	97.9%	100%	98.7%	0.31%	0.11
AllBorderCombinations											
Portfolio	min	3.63	50.9	2.22	73%	23%	100%	100%	100%	3.74%	4.826
	avg	3.64	51.33	2.25	73%	24%	100%	100%	100%	3.76%	10.908
	max	3.65	51.68	2.28	74%	24%	100%	100%	100%	3.77%	18.692
SPC	min	6.75	80.09	3.07	56%	15%	100%	100%	100%	2.64%	0.001
	avg	6.81	81.1	3.18	56%	16%	100%	100%	100%	2.69%	0.002
	max	6.85	82.18	3.24	57%	16%	100%	100%	100%	2.72%	0.002
ANT	min	3.64	54.8	7.31	71%	23%	100%	100%	100%	3.66%	1.625
	avg	3.65	56.93	8.7	72%	23%	100%	100%	100%	3.69%	7.674
	max	3.67	58.98	10.31	72%	23%	100%	100%	100%	3.7%	14.8
AGA	min	4.3	61.87	4.16	68%	21%	100%	100%	100%	3.47%	2.466
	avg	4.33	63.07	4.45	68%	21%	100%	100%	100%	3.49%	3.23
	max	4.36	64.86	4.64	69%	22%	100%	100%	100%	3.52%	4.192
TR	min	7.24	82.02	2.62	54%	15%	100%	100%	100%	2.68%	0.002
	avg	7.26	82.25	2.63	54%	15%	100%	100%	100%	2.69%	0.002
	max	7.29	82.61	2.65	54%	15%	100%	100%	100%	2.7%	0.003
EachBorderOnce											
Portfolio	min	3.18	42.92	2.08	79%	26%	93.5%	100%	97.1%	3.82%	4.137
	avg	3.19	43.18	2.12	79%	26%	93.9%	100%	97.4%	3.83%	9.259
	max	3.21	43.52	2.15	79%	26%	94.2%	100%	97.8%	3.84%	15.472
SPC	min	5.19	59.18	3.08	64%	18%	85.9%	100%	89.9%	2.79%	0.001
	avg	5.25	60.09	3.19	64%	19%	86.5%	100%	90.4%	2.83%	0.001
	max	5.27	60.67	3.24	65%	19%	87%	100%	91%	2.86%	0.001
ANT	min	3.19	47.06	6.13	75%	25%	90.7%	100%	95%	3.72%	1.392
	avg	3.21	47.78	6.55	76%	25%	91.1%	100%	95.2%	3.74%	6.697
	max	3.23	49.45	7.28	76%	25%	91.5%	100%	95.7%	3.77%	13.091
AGA	min	4.02	52.0	3.87	71%	22%	86%	100%	89%	3.22%	1.935
	avg	4.05	52.89	4.1	71%	22%	86.6%	100%	89.7%	3.25%	2.558
	max	4.09	53.87	4.22	72%	22%	87.2%	100%	90.3%	3.29%	3.365
TR	min	6.03	64.95	2.49	59%	17%	84.9%	100%	88.6%	2.69%	0.002
	avg	6.08	65.42	2.5	59%	17%	85.2%	100%	88.9%	2.7%	0.002
	max	6.1	65.79	2.52	59%	18%	85.4%	100%	89%	2.71%	0.003

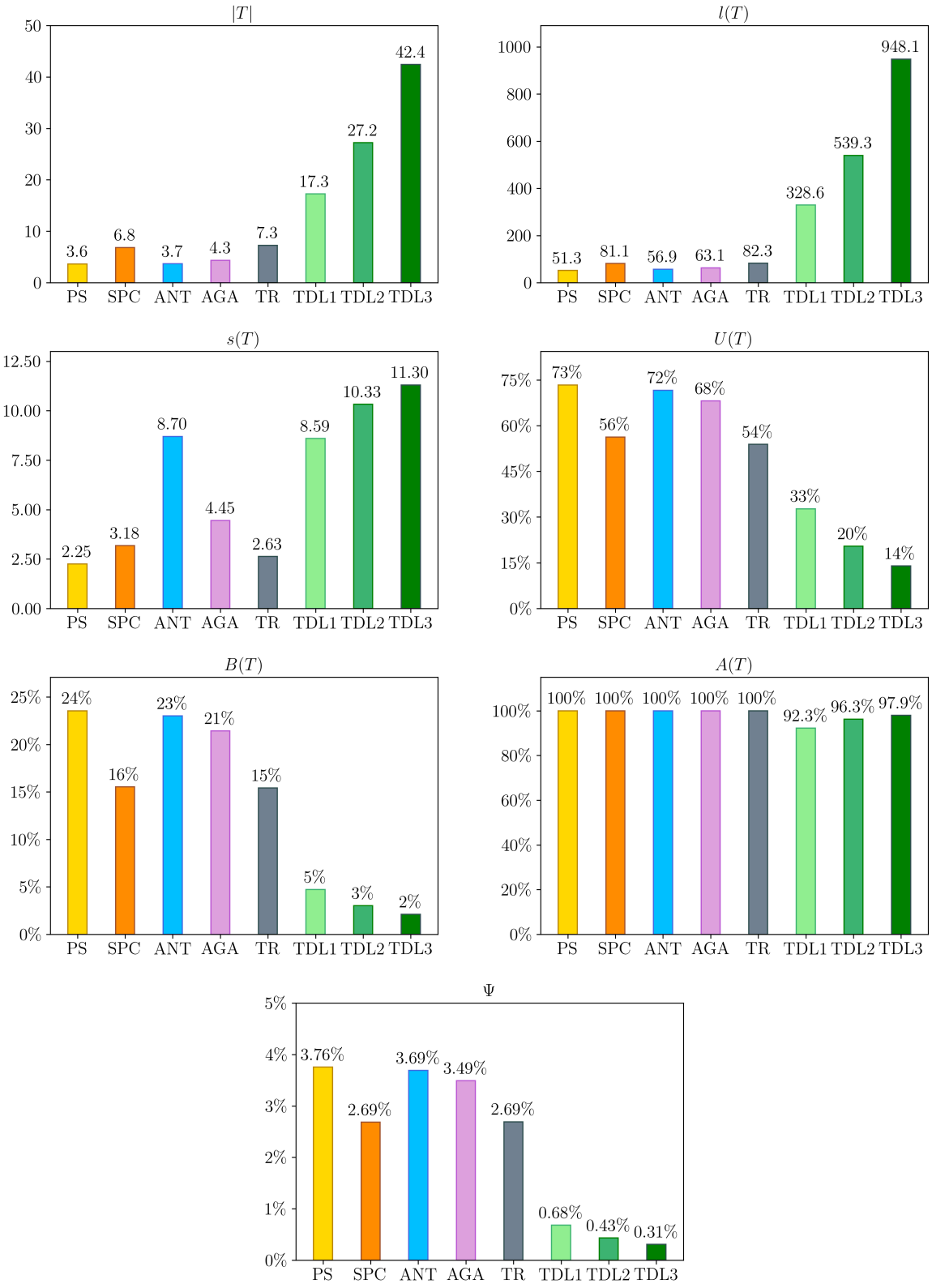


Figure 8.1: Algorithm comparison for *AllBorderCombinations* test coverage criterion through the test set evaluation criteria.

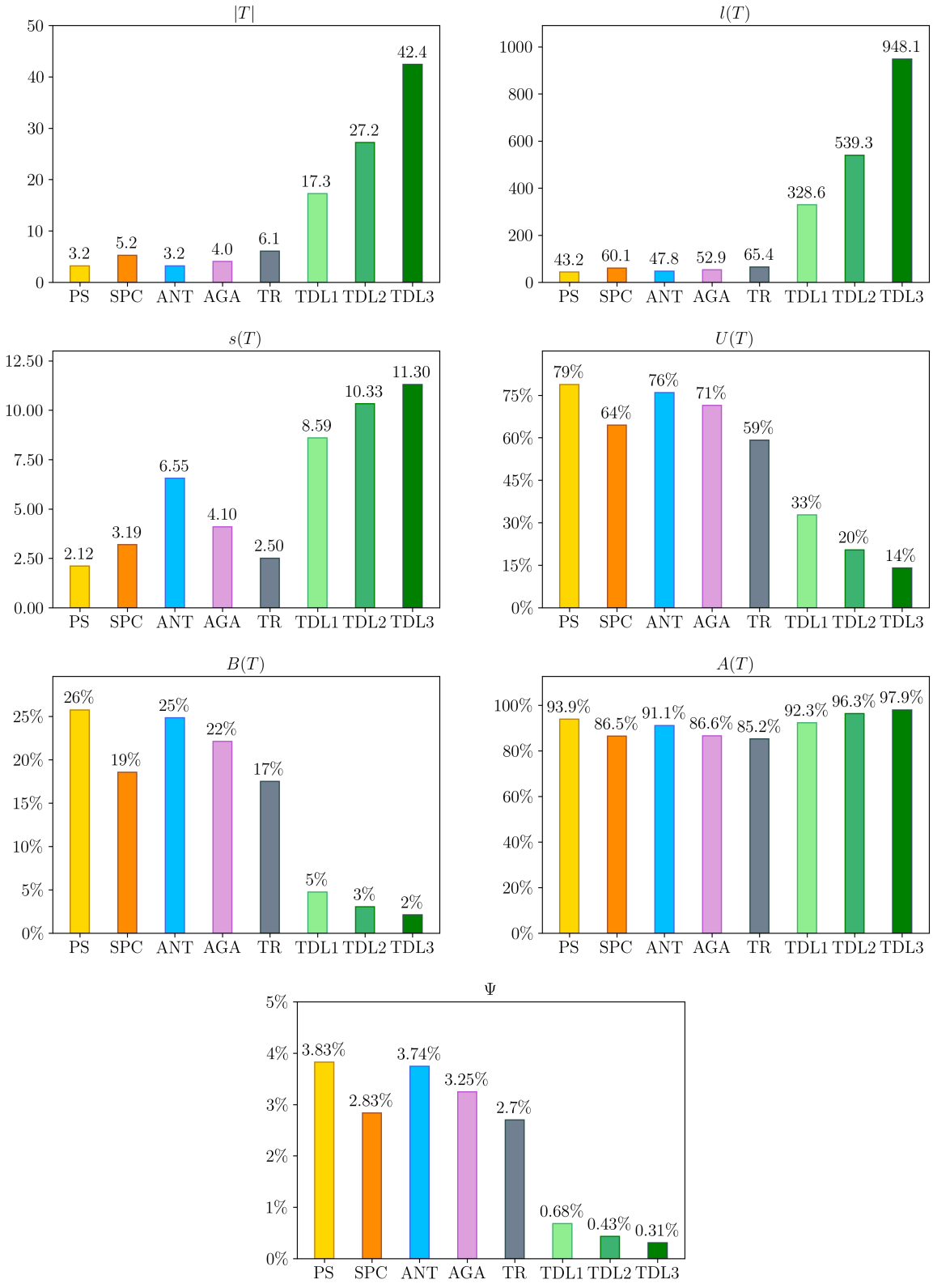


Figure 8.2: Algorithm comparison for *EachBorderOnce* test coverage criterion through the test set evaluation criteria.

average $l(T)$ was 9.7% higher than for the ANT algorithm. For the SPC algorithm, the average $l(T)$ was 20.5% higher than for the ANT algorithm, and for the TR algorithm, the average $l(T)$ was 27% higher than for the ANT algorithm.

For the average ratio of unique edges $U(T)$ contained in T, the situation for the *EachBorderOnce* coverage criterion was, again, similar to those for the *AllBorderCombinations* coverage, with the ANT algorithm visiting the highest number of unique edges, on average, and the TR algorithm the lowest. On average, T produced by the ANT algorithm had 7% higher $U(T)$ than this criterion for test sets produced by the AGA, 19% higher $U(T)$ than for the SPC algorithm, and 29% higher $U(T)$ than the results produced by the TR algorithm.

The ANT algorithm computed T with the highest average $B(T)$ as well: 14% higher than the same criterion for test sets produced by the AGA, 32% higher than $B(T)$ for the SPC algorithm, and 47% higher than for the TR algorithm.

For the length dispersion $s(T)$, the test sets produced by the ANT algorithm to satisfy the *EachBorderOnce* coverage criteria also had, on average, the highest value: 2.6 times higher than the average $s(T)$ of test sets produced by the TR algorithm, which had the best value for this criterion. The test sets produced by the SPC algorithm had an average $s(T)$ almost 28% higher than results produced by the TR algorithm, and the AGA produced test sets with an average $s(T)$ almost 64% higher than $s(T)$ of T produced by the TR algorithm.

Table 8.2: Average values of test set evaluation criteria over all G for the *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* test coverage criteria.

Criterion / alg		$\|T\|$	$l(T)$	$s(T)$	$U(T)$	$B(T)$	$A(T)$	$E(T)$	$\psi(T)$	Ψ	$t[s]$
both **ComprehensiveAllBorderCombinations** and **ComprehensiveEachBorderOnce**											
Edge		17.25	328.62	8.59	33%	5%	92.3%	100%	95.4%	0.68%	0.016
Edge-pair		27.19	539.27	10.33	20%	3%	96.3%	100%	97.6%	0.43%	0.041
TDL 3		42.44	948.07	11.3	14%	2%	97.9%	100%	98.7%	0.31%	0.11
ComprehensiveAllBorderCombinations											
Portfolio	min	7.22	97.07	2.99	54%	14%	100%	100%	100%	2.4%	11.421
	avg	7.22	97.37	3.04	54%	14%	100%	100%	100%	2.41%	26.593
	max	7.23	97.72	3.11	54%	14%	100%	100%	100%	2.41%	60.81
SPC	min	8.48	102.26	3.18	49%	13%	100%	100%	100%	2.24%	0.001
	avg	8.52	102.53	3.25	49%	13%	100%	100%	100%	2.26%	0.002
	max	8.56	102.85	3.32	49%	13%	100%	100%	100%	2.28%	0.007
ANT	min	7.23	109.52	9.87	52%	13%	100%	100%	100%	2.36%	3.101
	avg	7.24	112.62	11.19	52%	13%	100%	100%	100%	2.37%	19.554
	max	7.25	118.29	13.74	52%	13%	100%	100%	100%	2.38%	54.558
AGA	min	7.98	115.2	4.61	51%	13%	100%	100%	100%	2.25%	5.234
	avg	8.02	116.46	4.73	51%	13%	100%	100%	100%	2.26%	7.037
	max	8.08	117.81	4.86	51%	13%	100%	100%	100%	2.28%	8.51
ComprehensiveEachBorderOnce											
Portfolio	min	4.47	59.41	2.77	67%	18%	88.7%	100%	95.2%	2.74%	5.681
	avg	4.5	59.72	2.84	67%	18%	88.9%	100%	95.4%	2.76%	17.456
	max	4.52	60.09	2.94	68%	18%	89.5%	100%	95.7%	2.77%	41.986
SPC	min	6.01	69.34	3.15	58%	16%	87.1%	100%	90.6%	2.43%	0.001
	avg	6.04	69.76	3.27	59%	16%	87.5%	100%	91.3%	2.46%	0.001
	max	6.09	70.16	3.4	59%	16%	88.1%	100%	92.1%	2.49%	0.004
ANT	min	4.51	66.64	6.69	64%	17%	80.9%	100%	91.1%	2.66%	1.829
	avg	4.52	67.41	7.16	64%	17%	81%	100%	91.3%	2.67%	13.442
	max	4.54	68.14	7.64	65%	18%	81.2%	100%	91.6%	2.68%	35.129
AGA	min	5.26	69.19	4.11	63%	16%	81.1%	100%	87.9%	2.44%	2.75
	avg	5.3	70.47	4.3	63%	17%	81.4%	100%	88.4%	2.47%	4.013
	max	5.33	71.15	4.53	63%	17%	81.8%	100%	89.2%	2.5%	6.854

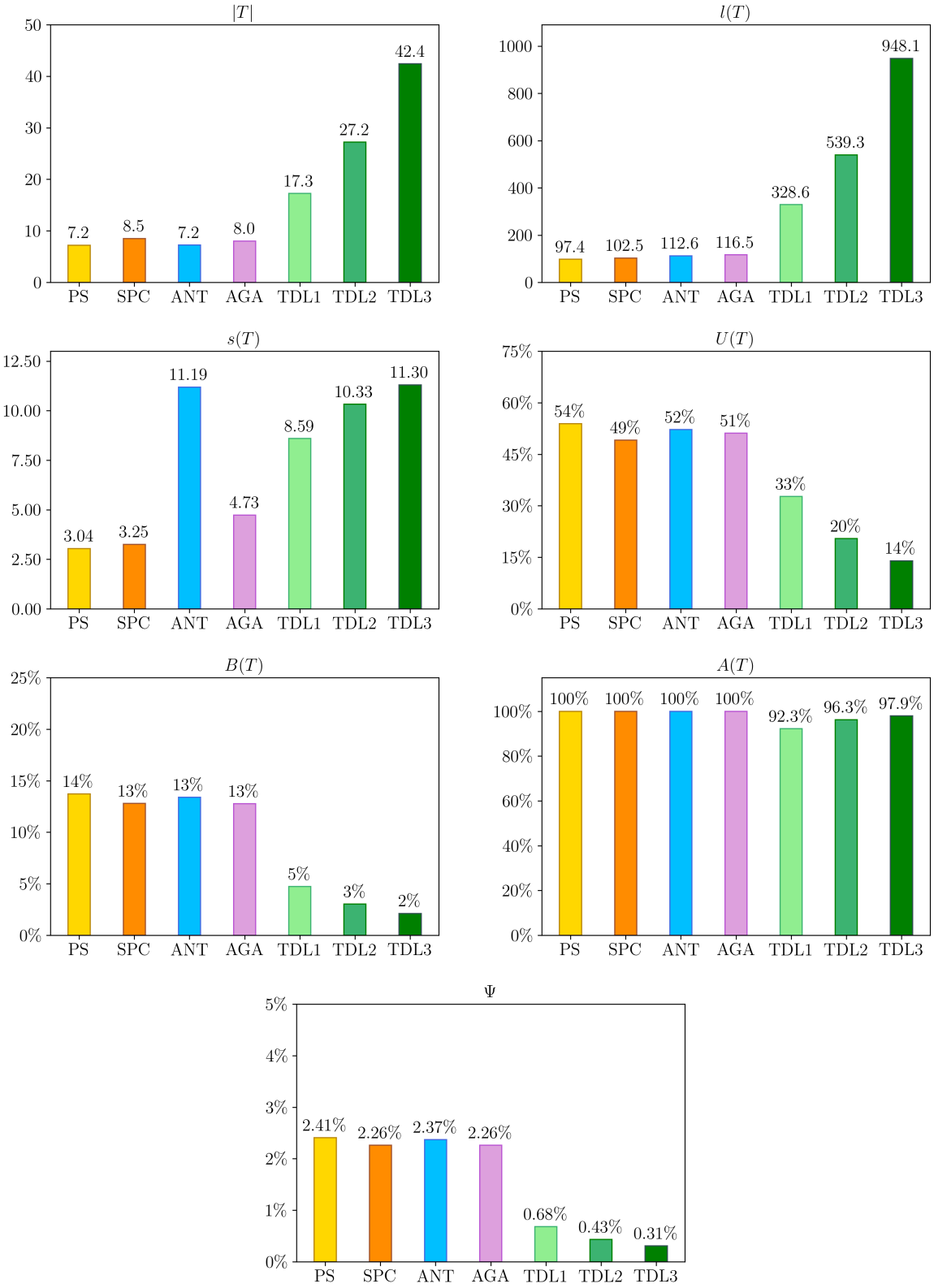


Figure 8.3: Algorithm comparison for *ComprehensiveAllBorderCombinations* test coverage criterion through the test set evaluation criteria.

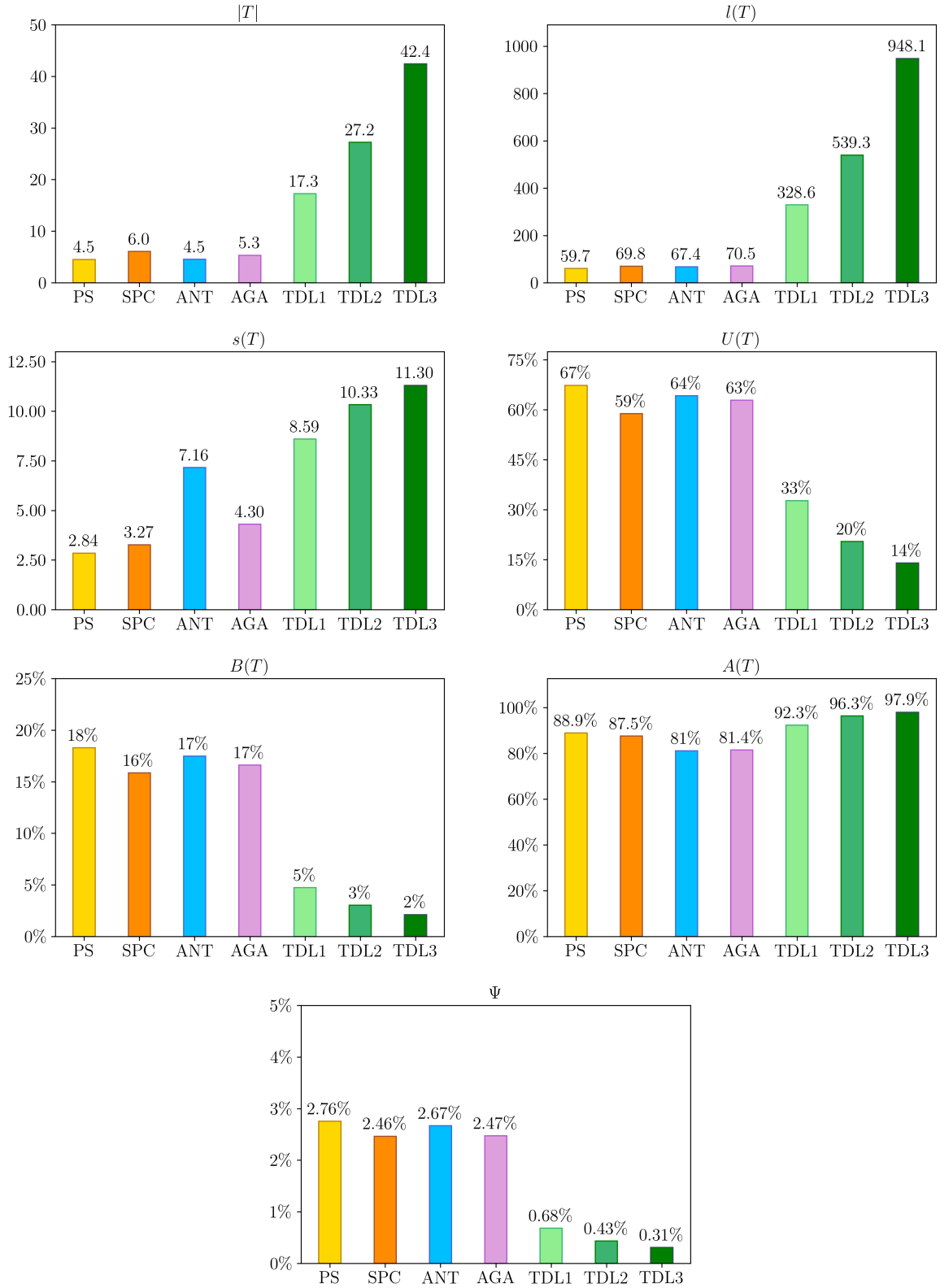


Figure 8.4: Algorithm comparison for *ComprehensiveEachBorderOnce* test coverage criterion through the test set evaluation criteria.

For the ***ComprehensiveAllBorderCombinations*** test coverage criterion, all relevant algorithms (SPC, ANT, and AGA) yielded similar results in the average $|T|$ criterion. The ANT algorithm's value of average $|T|$ was 15% lower than the result of the same criterion for the SPC algorithm and 10% lower than the result for the AGA (see Table 8.2 and Figure 8.3).

There was a different outcome for the $l(T)$ criterion, for which, on average, the SPC algorithm produced T with the best result. Then, the ANT algorithm generated test sets with a 9% higher value of $l(T)$ than the SPC algorithm, and the AGA generated test sets with $l(T)$ 12% higher than the SPC algorithm.

For the $U(T)$ criterion, all the algorithms achieved similar results, with the ANT algorithm computing T having the highest value of this criterion, which was only 2% lower than the value of $U(T)$ for the AGA, and 6% lower than the SPC algorithm.

For the $B(T)$ criterion, the difference between test sets produced by the individual algorithms was negligible.

On the other hand, for the $s(T)$ criterion, the SPC algorithm returned test sets with the best average value. The AGA computed T with an average $s(T)$ 46% higher than the SPC algorithm, and the ANT algorithm with a 3.5 times greater average value of $s(T)$ than the SPC algorithm.

For the ***ComprehensiveEachBorderOnce*** test coverage criterion, the average size of $|T|$ computed by the ANT algorithm was 15% lower than for the AGA and 25% lower than for the SPC algorithm. The results are presented in Table 8.2 and visualized in Figure 8.4.

Regarding the average total length of the test set $l(T)$, all the algorithms produced test sets with similar average values of this criterion. The SPC algorithm computed test sets with an average value of $l(T)$ 3.5% bigger than the test sets computed by the ANT algorithm, and the test sets computed by the AGA algorithm had an average $l(T)$ 4.5% bigger than those computed by the ANT algorithm.

For the $U(T)$ criterion and *ComprehensiveEachBorderOnce*, the results were very similar to the *ComprehensiveAllBorderCombinations* results. The AGA computed test sets that had an average value of this criterion 2% higher than those computed by the ANT algorithm, and the SPC algorithm computed test sets that had a value 8% higher than those computed by the ANT algorithm.

The average number of border nodes in the test set $B(T)$ was the same for the test sets computed by the ANT algorithm and the AGA, and 6% higher than for the SPC algorithm.

Considering the average length dispersion $s(T)$, test sets computed by the AGA had a value of this criterion that was 31% higher than those by the SPC algorithm. For the

ANT algorithm, the value was 2.19 times higher than for the SPC algorithm.

The SPC, ANT, AGA, and TR algorithms outperformed all test coverage criteria in the initial baseline in the number of test steps $l(T)$, the differences being approximately threefold for *Edge* coverage and AGA for *ComprehensiveAllBorderCombinations* up to the values differing approximately 20 times for *TDL 3* and ANT for *EachBorderOnce*. Regarding other $\mathcal{E}$, the differences were similarly significant (details can be found in Tables 8.1 and 8.2). Hence, the results of the initial baseline confirm that a standard path-based testing approach using *Edge*, *Edge-pair* or *TDL 3* criteria is not suitable for generating T for the limited network connectivity testing problem as discussed in this thesis.

8.2 Algorithms that Produced the Best Test Sets for Particular SUT Models

The averaged properties of T produced by individual algorithms, as discussed in Section 8.1, give helpful insight into the algorithms' performances. Still, it is only one of the possible viewpoints on the results. Another question is, which algorithm computed T with the best result (for particular $\mathcal{E}$) for the majority of the SUT models?

First, the results for the *EachBorderOnce* and *AllBorderCombinations* criteria are presented in Figure 8.5, followed by the results for the *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations* presented in Figure 8.6.

The charts in Figures 8.5 and 8.6 use the y-axis to present the individual test set evaluation criteria introduced in Sections 4.3, 5.1, and 7.3. The x-axis presents the individual algorithms. Each number in the intersection of the axes represents the number of cases in which a particular algorithm returned the best T for a particular evaluation criterion. The size of the bubble around a number visually emphasizes that number. For the comparisons of the evaluation criteria results, either a comparator $<$ or $>$ was used, based on the meaning of particular $\mathcal{E}$ (see Sections 4.3, 5.1, and 7.3). In these comparisons, only the situations when particular algorithm clearly outperformed the others were considered; if the best result of an evaluation criterion for a SUT model G corresponded to T computed by more than one algorithm, then no algorithm was considered as the best for the particular model.

As the SPC, ANT, AGA, and TR algorithms are non-deterministic and the computations have been run ten times, Figures 8.5 and 8.6 present the average results. On the right side of the bubbles, the numbers in brackets represent the worst cases and the best cases of all these ten runs.

As the TR algorithm is not applicable as a baseline for the *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations* criteria (the reason was explained in

Section 5.2), it is not present in Figure 8.6.

Figure 8.5a shows that the ANT algorithm outperformed the others for the ***AllBorderCombinations*** coverage in many criteria. Specifically, it gave the best result for 116 SUT models for $|T|$, for 140 models for $l(T)$, for 128 models for $U(T)$, and for 145 models for $B(T)$.

The ANT algorithm is followed by the second-best performer, the AGA, which provided the best result for 45 SUT models for $l(T)$, 82 models for $U(T)$, and 44 models for $B(T)$.

Moreover, for a small number of SUT models, it was the SPC algorithm that provided the best solution (six models for $l(T)$, $U(T)$, and $B(T)$). The TR baseline produced the best solution for a very low number of models (two models for $l(T)$, and one model for $B(T)$).

For $s(T)$, the best results were provided by the TR baseline (114 SUT models), followed by the SPC algorithm (36 models).

For the ***EachBorderOnce*** coverage (see Figure 8.5b), the outcome was similar to the previous test coverage criterion. The ANT algorithm outperformed the others for 154 models for the $|T|$, 138 models for $l(T)$, 130 models for $U(T)$, and 145 models for $B(T)$. On the other hand, the ANT results were not as good for the $s(T)$ criterion, for which this algorithm provided the best result for 22 of the models only.

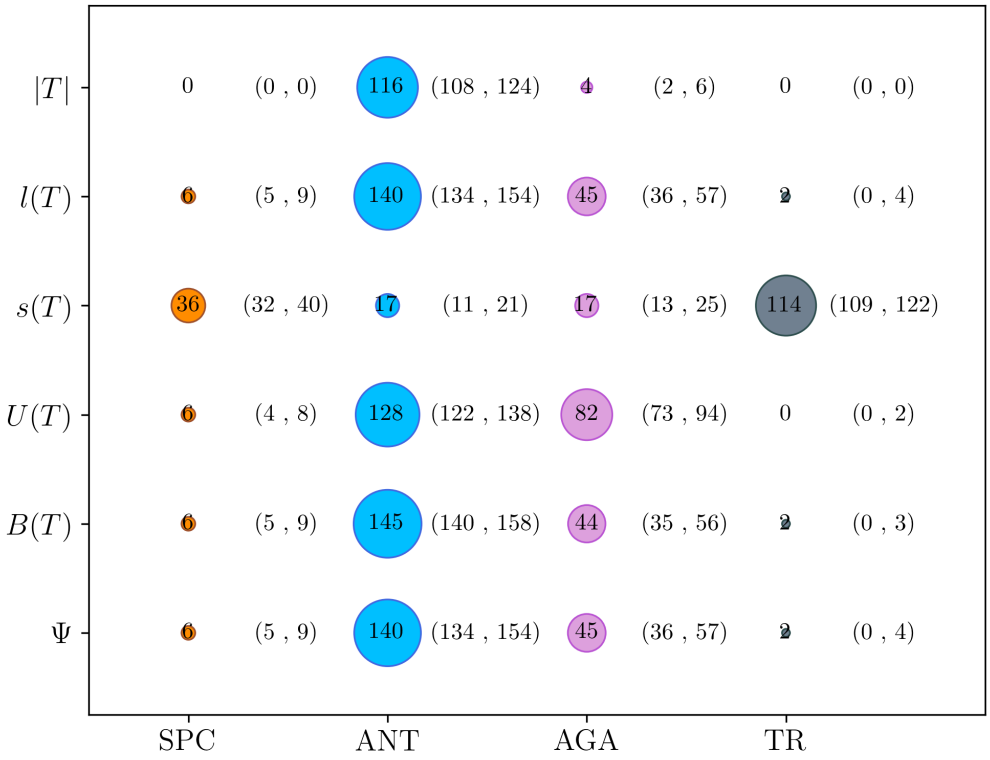
The AGA was the second-best performer, yielding the best results for 57 models for $l(T)$, 81 models for $U(T)$, and 56 models for $B(T)$. For the *EachBorderOnce*, the number of SUT models, for which the SPC provided the best result, was generally higher compared to *AllBorderCombinations*. The SPC algorithm gave the best result for 17 SUT models for $l(T)$, 18 models for $U(T)$, and 15 models for $B(T)$.

The TR baseline was the best performer for $s(T)$. It provided the best result for 107 SUT models, followed by the SPC algorithm, which provided the best result for 33 models.

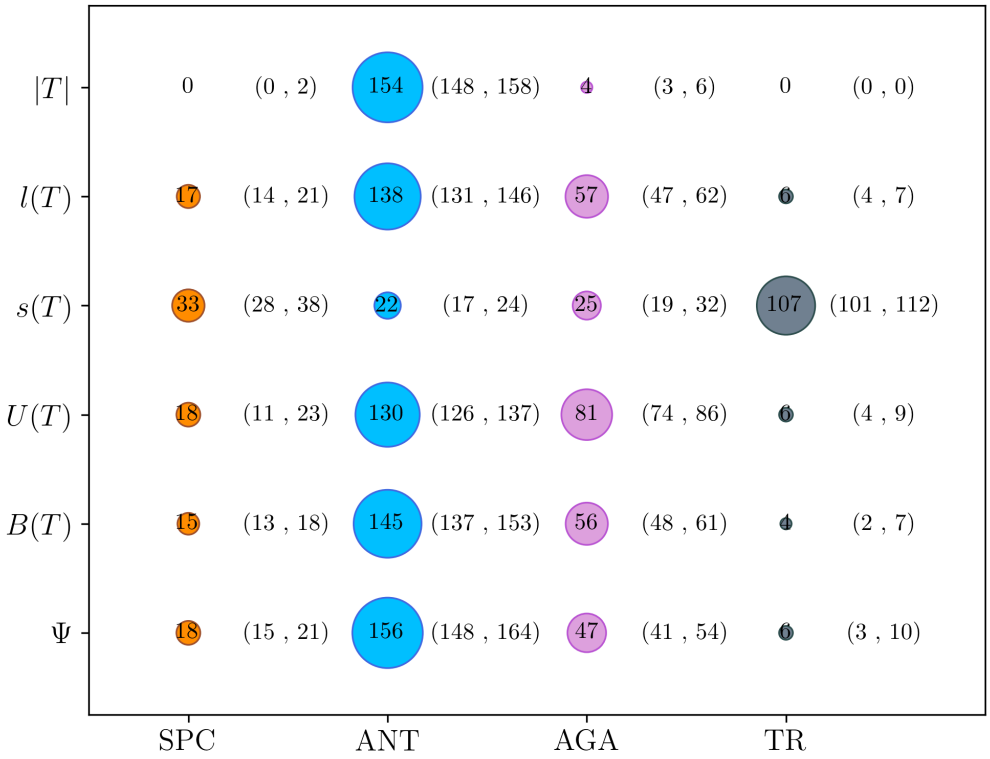
For *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* (see Figure 8.6), the results were more diverse. For both criteria, the ANT algorithm was still the best performer. Nevertheless, the difference between its performance and the performances of the AGA and the SPC algorithm was not as large as it was for the previous *AllBorderCombinations* and *EachBorderOnce* criteria.

For the ***ComprehensiveAllBorderCombinations*** criterion, results depicted in Figure 8.6a, the ANT algorithm computed the best solution for 96 SUT models for $|T|$, for 86 models for $l(T)$, for 113 models for $U(T)$, and 93 for $B(T)$.

The AGA was the second-best performer for $U(T)$ (84 SUT models). For the rest of the test set evaluation criteria, the second place was secured by the SPC algorithm (58

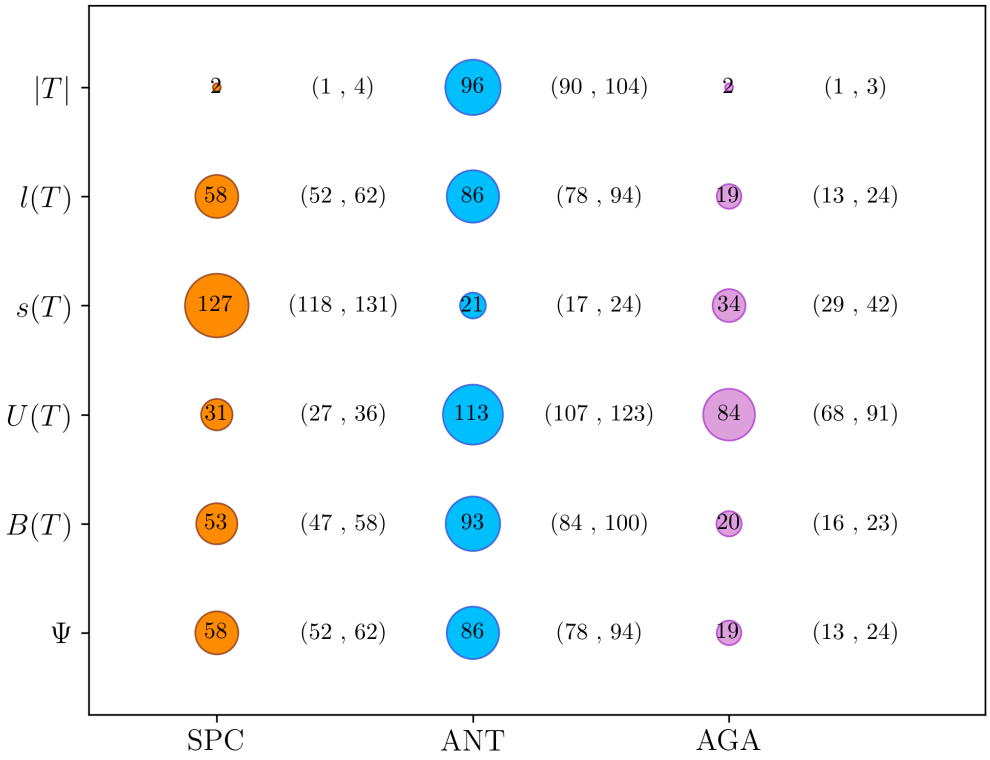


(a) Comparison for the *AllBorderCombinations* coverage criterion.

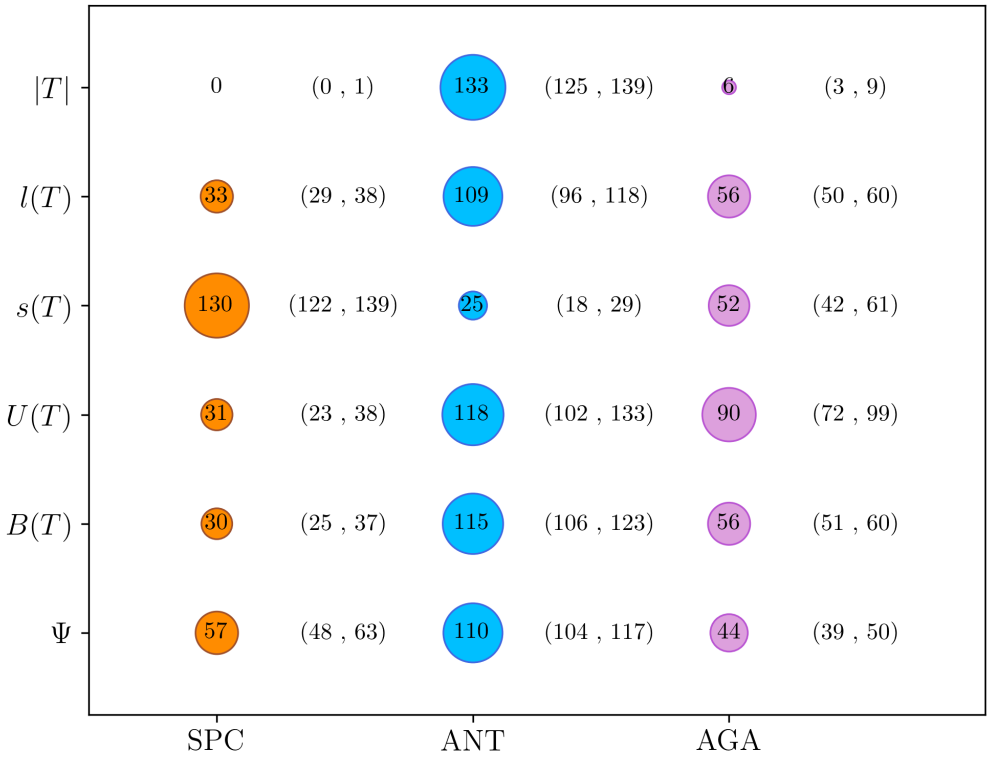


(b) Comparison for the *EachBorderOnce* coverage criterion.

Figure 8.5: The visualization of the number of SUT models for which the individual algorithms produced the best T (part 1).



(a) Comparison for the *ComprehensiveAllBorderCombinations* coverage criterion.



(b) Comparison for the *ComprehensiveEachBorderOnce* coverage criterion.

Figure 8.6: The visualization of the number of SUT models for which the individual algorithms produced the best T (part 2).

models for $l(T)$, 127 models for $s(T)$, and 53 models for $B(T)$). However, in third place for performance was the remaining algorithm with a relatively significant number of SUT models for each of the test set evaluation criteria.

For the ***ComprehensiveEachBorderOnce***, the ANT algorithm outperformed its competitors in $l(T)$ more clearly in comparison to *ComprehensiveAllBorderCombinations*; it provided the best result for 133 models for $|T|$, for 109 models for $l(T)$, for 118 models for $U(T)$ and 115 models for $B(T)$. The second-best performer for this criterion was the AGA, yielding the best result for 56 models for $l(T)$, 90 models for $U(T)$, and 56 models for $B(T)$. The SPC algorithm performed best for 130 models for $s(T)$, while still maintaining a considerably high number of SUT models–in which it provided the best results. However, in overall comparison, SPC was the worst performer for the *ComprehensiveEachBorderOnce* criterion.

As a preliminary summary, the ANT algorithm was the best performer in the conducted experiments, followed by the AGA and the SPC algorithm. However, there was no "superior algorithm" that provided the best solution for all SUT models universally. We are going to draw further conclusions from these findings later in Section 9.

8.3 Effectiveness of Limited Network Connectivity Defects Detection in the SUT

A simple type of limited network connectivity-related defects that can be activated by visiting a single LCZ IN or LCZ OUT node will be all activated by T, satisfying any of *AllBorderCombinations*, *EachBorderOnce*, *ComprehesiveAllBorderCombinations* , or *ComprehensiveEachBorderOnce* criteria.

Hence, in this section, we analyze the potential of T in detecting the second, more complex variant of the limited network connectivity-related defects in the SUT, $\chi(G)$.

A test set T satisfying the *AllBorderCombinations* coverage criterion activates 100% of the defined defect node pairs $\chi(G)$, which is a consequence of the *AllBorderCombinations* definition. In the same way, T satisfying *ComprehensiveAllBorderCombinations* activates 100% of the defect node pairs defined in G.

The results for *EachBorderOnce* and *ComprehensiveEachBorderOnce* are worth a more detailed analysis (values of $\psi(T)$ in Tables 8.1 and 8.2).

In the experiments, T generated by the TR baseline, satisfying *EachBorderOnce*, detected 88.9% of the defect node pairs on average for all SUT models and computation runs. T generated by the AGA detected 89.7%, T generated by the SPC algorithm detected 90.4%, and T generated by the ANT algorithm detected 95.2% of these defects.

For *ComprehesiveEachBorderOnce*, T generated by the AGA detected 88.4% of the

defect node pairs, on average, and T generated by the ANT and SPC algorithms detected 91.3% of these defects.

However, these results have to be put in the context of T size, $l(T)$. Hence, Ψ is the most relevant indicator to evaluate and should be analyzed for all test coverage criteria.

As explained before, note that values of Ψ presented in Tables 8.1 and 8.2 are not equal to the division of the values of average ψ over average $l(T)$ as presented in these tables, but they are an average of $\frac{\psi}{l(T)}$ for each of the SUT models.

For *AllBorderCombinations*, the best value of Ψ was achieved by the ANT algorithm (3.69%), followed by the AGA (3.49%), and then the SPC and TR baseline algorithms (both with 2.69%).

For *EachBordeOnce*, the ANT algorithm yielded the test sets with Ψ being 3.74%, followed by the AGA with 3.25%, the SPC algorithm with 2.83%, and the TR baseline algorithm with 2.7%.

In the case of *ComprehesiveAllBorderCombinations* and *ComprehesiveEachBorderOnce*, the differences between Ψ for the individual algorithms were much smaller (due to $\psi(T)$ being 100% for these two test coverage criteria). For *ComprehesiveAllBorderCombinations*, the best value was provided by the ANT algorithm (2.37%), followed by 2.26% in the case of the SPC algorithm and the AGA. For *ComprehesiveEachBorderOnce*, the best value (2.67%) was provided by the ANT algorithm again, followed by the SPC algorithm (2.46%), and the AGA (2.47%).

For the initial baselines, values of Ψ were much lower, 0.68% for *Edge*, 0.43% for *Edge-pair*, and 0.31% for *TDL 3* coverage criteria, which rendered the approach that was suggested by the initial baseline significantly ineffective.

8.4 Results of the Portfolio Strategy

The results of the portfolio strategy are presented in Table 8.1 for the *EachBorderOnce* and *AllBorderCombinations* criteria, and then in Table 8.2 for *ComprehensiveEachBorderOnce* and *ComprehensiveAllBorderCombinations* criteria.

Also, the data are presented in Figure 8.1 for *AllBorderCombinations*, in Figure 8.2 for *EachBorderOnce*, in Figure 8.3 for *ComprehensiveAllBorderCombinations*, and in Figure 8.4 for the *ComprehensiveEachBorderOnce* criterion, where the results for the portfolio strategy are denoted as PS (the first yellow column in the graphs).

As the portfolio strategy selects the best T from the individual results provided by the SPC, ANT, AGA, and TR algorithms, it is not included in the overall statistics presented in Figures 8.5 and 8.6.

In this section, we analyze the differences between the results of the portfolio strat-

egy and the best-performing algorithm, both running separately for the set of 310 SUT problem models.

Starting with $|T|$, the portfolio strategy gave practically the same result as the ANT algorithm, the best performer for this test set comparison criterion for all test coverage criteria, the differences being under 0.6% in favor of the portfolio strategy for all test coverage criteria.

The more important indicator is $l(T)$, being a proxy of testing costs. The portfolio strategy selected T with a number of steps ($l(T)$) lower by 9.8% than the ANT, the best performer for the *AllBorderCombinations* criterion, very similarly, lower than 9.6% than the ANT algorithm for the *EachBorderOnce* criterion, lower by 5.0% than SPC algorithm, the best performer for the *ComprehensiveAllBorderCombinations* criterion, and lower by 11.4% than the ANT, the best performer for the *ComprehensiveEachBorderOnce* criterion.

Regarding the test cases length dispersion $s(T)$, the result of the portfolio strategy was lower by 14.4% than the result for the best performing TR baseline for the *AllBorderCombinations* criterion, and by 15.2% for the *EachBorderOnce* criterion. For *ComprehensiveAllBorderCombinations*, the result of the portfolio strategy was 6.5% lower than the SPC algorithm, the best performer for this criterion. For *ComprehensiveEachBorderOnce*, the portfolio strategy selected T with $s(T)$ lower by 13.1% than the SPC algorithm.

In the case of potential edge redundancy $U(T)$, the ANT algorithm computed T with the best values of $U(T)$ for all test coverage criteria. In comparison to the ANT algorithm, the portfolio strategy selected T with $U(T)$ better by 1.4% for *AllBorderCombinations*, better by 3.9% for *EachBorderOnce*, better by 3.8% for *ComprehensiveAllBorderCombinations* and better by 4.7% for the *ComprehensiveEachBorderOnce* criterion. These results are not as significant as those in the cases of $l(T)$ and $s(T)$.

The next indicator is the effectiveness of visiting LCZ border nodes $B(T)$. Compared to the ANT algorithm, the portfolio strategy selected T with $B(T)$ better by 4.3% for *AllBorderCombinations*, and by 4.0% for *EachBorderOnce*. For $B(T)$ and *ComprehensiveAllBorderCombinations*, all three AGA, SPC, and ANT algorithms performed well, having the value of $B(T)$ as approximately 13%; however, the portfolio strategy achieved a value higher by 7.7%. For $B(T)$ and *ComprehensiveEachBorderOnce*, both the ANT algorithm and the AGA yielded the best results, 17%, but the portfolio strategy achieved a value of $B(T)$ higher by 5.9%.

For $A(T)$, only the *EachBorderOnce* and *ComprehensiveEachBorderOnce* criteria are relevant to evaluate, and the differences between the results of the portfolio strategy and the best-performing algorithm are not that significant. For *EachBorderOnce*, the portfolio strategy produced T with $A(T)$ higher by 3.1% than T produced by ANT, the best performer for this criterion. For *ComprehensiveEachBorderOnce*, the $A(T)$ of the

test set selected by the portfolio was higher by 1.6% than of the test set computed by SCP algorithm, the best SPC for this test coverage criterion.

The last important aspect to analyze is the probability of T to detect artificial defects in the SUT. Here, for *AllBorderCombinations* and *ComprehensiveAllBorderCombinations*, the portfolio strategy produced T that detected 100% of defect node pairs $\chi(G)$ defined in G. For *EachBorderOnce*, on average, the portfolio strategy produced T that activated 97.4% of the defect node pairs defined in individual G in the experiments, the best result compared to the individual algorithms.

For *ComprehensiveEachBorderOnce*, the portfolio strategy produced T that, on average, activated 95.4% of the defect node pairs defined in individual G.

With Ψ as the major indicator of artificial defect detection effectiveness, for *AllBorderCombinations*, the average Ψ for the portfolio strategy was 3.8%. This result was 1.9% higher than for the ANT algorithm, 7.7% higher than for the AGA, and more than 39.8% higher than for both the SPC algorithm and the TR baseline, which achieved the same value of the average Ψ.

For *EachBorderOnce*, the average Ψ for the portfolio strategy was 3.8%, which was 2.4% higher than for the ANT algorithm, 17.8% higher than for the AGA, 35.3% higher than for the SPC algorithm, and 41.9% higher than for the TR baseline.

For *ComprehensiveAllBorderCombinations*, the Ψ was 2.4% for the portfolio strategy, 1.7% higher than for the ANT algorithm, and 6.6% higher than the Ψ was for both the AGA and the SPC algorithm.

Finally, for the *ComprehensiveEachBorderOnce*, the average Ψ for the portfolio strategy was 2.8%, a value that was 3.4% higher than for the ANT algorithm, 11.7% than for the AGA, and 12.2% higher than the Ψ was for the SPC algorithm.

Regarding the initial baselines, the differences between the averaged Ψ for the portfolio strategy and averaged Ψ for *Edge*, *Edge-pair*, and *TDL 3* were more significant (see Tables 8.1 and 8.2). These results clearly demonstrate that, for the limited network connectivity testing problem being the subject of this thesis, the proposed approach outperforms the available path-based testing alternatives examined in the initial baseline.

8.5 Time Effectiveness of the Algorithms

In the $t[s]$ column, Table 8.1 presents average run times of compared algorithms in seconds for the *AllBorderCombinations* and *EachBorderOnce* test coverage criteria.

For ***AllBorderCombinations***, both the SPC and TR algorithms returned the results with the shortest run time of 0.002 s. The AGA took much longer to compute T, taking 3.3 s on average. The slowest was the ANT algorithm, which, on average, needed 7.7 s.

The longest run time measured during the experiments for the *AllBorderCombinations* criterion was 204.8 s for the AGA (for SUT model id=98, $|N|$=93 and $|E|$=170, see Table 7.3), followed by 57.2 s for the ANT algorithm (SUT model id=290, $|N|$=81 and $|E|$=113, see Table 7.4). For the largest SUT model, in terms of both the number of nodes and edges out of all instances ($|N|$=325 and $|E|$=488, id=147 in Table 7.3), the longest run times out of all ten repetitions were 106.4 s for the AGA, followed by 5.4 s for the ANT algorithm, 0.018 s for the TR algorithm, and 0.002 s for the SPC algorithm.

The trend for the average run times of the algorithms was similar for both the *EachBorderOnce* test coverage criterion and *AllBorderCombinations*. For ***EachBorderOnce***, the SPC algorithm was the fastest with an average run time of 0.001 s, followed by the TR baseline with 0.002 s. The AGA ran on all instances and all ten repetitions for 2.6 s on average, and the slowest was the ANT algorithm, which ran for 6.7 s on average.

For the *EachBorderOnce*, the longest run time measured during the experiments was 231.5 s for the AGA (for SUT model id=127, $|N|$=93 and $|E|$=149, see Table 7.3), followed by 57.7 s for the ANT algorithm (SUT model id=284, $|N|$=62 and $|E|$=77, see Table 7.4). For the largest SUT model, in terms of both the number of nodes and edges out of all instances ($|N|$=325 and $|E|$=488, id=147 in Table 7.3), the longest run times out of all repetitions were 22.2 s for the AGA, followed by 5.5 s for the ANT algorithm, 0.021 s for the TR algorithm, and 0.001 s for the SPC algorithm.

In the case of *EachBorderOnce*, the run times were generally slightly shorter than for *AllBorderCombinations*, which is a logical and expected result considering the definitions of these test coverage criteria.

Table 8.2 presents the average run times of the algorithms compared for the *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* criteria. Generally speaking, the trends are similar to those observed for *AllBorderCombinations* and *EachBorderOnce*.

In the case of ***ComprehensiveAllBorderCombinations*** criterion, the fastest was the SPC algorithm, which ran for all SUT models for 0.002 s on average. For the same task (averaged for all ten repetitions of computation), the AGA needed 7 s, followed by the ANT algorithm, the slowest, which, on average, needed 19.6 s to compute T.

For *ComprehensiveAllBorderCombinations*, the longest measured run time during the experiments was 289.8 s for the AGA (for SUT model id=142, $|N|$=271 and $|E|$=351, see Table 7.3), followed by 211.4 s for the ANT algorithm (SUT model id=270, $|N|$=79 and $|E|$=122, see Table 7.4). For the largest SUT model, in terms of both the number of nodes and edges out of all instances ($|N|$=325 and $|E|$=488, id=147 in Table 7.3), the longest run times out of all repetitions were 287.5 s for the AGA, followed by 9 s for the ANT algorithm, and 0.011 s for the SPC algorithm.

Finally, for the ***ComprehensiveEachBorderOnce*** criterion, the SPC algorithm was the fastest with an average run time of 0.001 s. To compute T, the AGA needed 4 s, and the slowest was the ANT algorithm, which ran for all SUT models and ten computations for 13.4 s on average.

When analyzing the SUT models with the longest run times and the run times for the largest SUT models for the *ComprehensiveEachBorderOnce* criterion, the longest run time measured during the experiments was 283.1 s for the AGA (for SUT model id=135, $|N|$=80 and $|E|$=122, see Table 7.3), followed by 100.1 s for the ANT algorithm (SUT model id=285, $|N|$=74 and $|E|$=97, see Table 7.4). For the largest SUT model, in terms of both the number of nodes and edges out of all instances ($|N|$=325 and $|E|$=488, id=147 in Table 7.3), the longest run times out of all repetitions were 30.5 s for the AGA, followed by 18.6 s for the ANT algorithm, and 0.002 s for the SPC algorithm.

Generally, the run times needed by the algorithms to satisfy *ComprehensiveAllBorderCombinations* were longer than those for *ComprehensiveEachBorderOnce*, which is in accordance with the principle of these test coverage criteria.

When looking at the initial baselines, no excessive run times were identified. The average run time for *Edge* coverage was 0.016 s, for *Edge-pair*, it was 0.041 s, and for *TDL 3*, it was 0.11 s.

We interpret and analyze the results later in Section 9. However, at this point, it is worth noting that even the relatively high average run times of the ANT algorithm (6.7 s up to 19.6 s for the particular test coverage criterion), which might cause user discomfort when computing the test cases, are partially a result of relatively large SUT models appearing in the experiments. When compared to the potential effort savings as a result of applying the proposed technique in an industrial project, for example, some slight user discomfort when generating the test cases would be considered acceptable.

Regarding the run times of the portfolio strategy, in the case of sequential execution of the algorithms, it was a sum of run times of all included algorithms. The average run times of the portfolio strategy varied from 9.3 s for the *EachBorderOnce* test coverage criterion to 26.6 s for the *ComprehensiveAllBorderCombinations* test coverage criterion, excluding the neglectable time to select the best T.

The previous note about the potential testing effort savings (especially for large SUT process models in complex IoT systems) making up for possible user discomfort when computing the test cases is also valid in this case. Moreover, the execution time of the portfolio strategy can be reduced by parallel execution of the individual employed algorithms.

Chapter 9

Discussion

The experimental results revealed that no algorithm is universally superior for obtaining the best T across all SUT models for all test set evaluation criteria. However, certain algorithms performed well in individual situations, as observed from the data.

For the *AllBorderCombinations* and *EachBorderOnce* coverage criteria, the ANT algorithm produced T with the best results for the number of test set evaluation criteria for the majority of SUT models (refer to Figure 8.5). The only exception was the length dispersion $s(T)$ in which the TR baseline demonstrated the best results. However, the AGA ranks the second-best performance and provided the best results in a considerable number of SUT models (approximately one-fourth to one-third on average, depending on $\mathcal{E}$).

For *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* coverage, the superiority of ANT over SPC and AGA was relatively lower compared to *AllBorderCombinations* and *EachBorderOnce* (Figure 8.6). ANT provided the best result for the majority of $\mathcal{E}$, except $s(T)$, and "the second place" varied for each test coverage criterion and $\mathcal{E}$. The same situation was observed for Ψ.

Therefore, to obtain the best T for a particular G, $\mathcal{E}$, and test coverage criterion, the SPC, ANT, and AGA algorithms (including the TR baseline for *AllBorderCombinations* and *EachBorderOnce* criteria) must be combined in a portfolio strategy.

The differences between the T generated by the portfolio strategy and the algorithm that achieved the best result for $|T|$, $U(T)$, and $B(T)$ were almost equal (less than 5% in most cases). Thus, we can conclude that the best-performing algorithms were well-designed for these criteria.

In contrast, the differences between the results of the portfolio strategy and the winning algorithms for all four coverage criteria was much more significant for $l(T)$ at approximately 10% (except *ComprehensiveAllBorderCombinations* with a difference of only 5%).

The difference was even larger for the length dispersion $s(T)$, which was approximately 15% for the *AllBorderCombinations* and *EachBorderOnce* criteria and 13% for the *ComprehensiveEachBorderOnce* criterion. Note that *ComprehensiveAllBorderCombinations* is an exception, and the difference in this test coverage criterion was only 6% approximately.

Finally, further improvements in artificial defect detection potential Ψ, up to approximately 4%, were observed upon comparing the portfolio strategy with the best-performing algorithm.

In summary, these results document further improvements in the average quality of T that can be achieved by employing the portfolio strategy.

Upon further analyzing the results of the individual algorithms, the ANT notably produced a lower number ($|T|$) of longer test cases with fewer steps in total ($l(T)$) compared to the other algorithms. This result corresponds well with the algorithm principle that ants attempt to visit more LCZs in a single test case. This effect also results in relatively high $s(T)$ for T computed using ANT. Notably, the SPC produced shorter test cases, but possibly with a greater number of total steps, because of the "greedy" nature of this algorithm for traversing G. In contrast to the ANT, the proximity of the individual LCZs was not considered in the SPC method.

In terms of run times, even the longest run time of the ANT algorithm in the entire experiment (289.8 s) can be considered acceptable. This run time was recorded for the SUT model with 271 nodes and 351 edges, which is a rare situation in industrial practice. In general, models with more than 100 nodes are expected to be less common. Moreover, even if an algorithm ran for minutes which might cause user's discomfort, we believe that the potential savings in test effort (which will increase for SUTs with such complexity) would undoubtedly compensate for such discomfort. The same principle applies to the run times of the portfolio strategy, which—in case of the sequential execution of the included algorithms—is a sum of the individual algorithm run times plus the time required to determine the best T by the selected $\mathcal{E}$. However, the execution period can be optimized by conducting parallel execution of the included algorithms.

Analyzing the longest run time versus that for the largest SUT model, the G size in terms of $|N|$ and $|E|$ evidently influenced the computation period; however, the graph topology combined with the presence of LCZs affects this duration even more. Interestingly, the SUT models corresponding to the longest computation period for T were the same for all algorithms and all test coverage criteria. Upon analyzing the principle of the algorithms, topology of SUT models, and experimental data, the number of cycles present in G significantly influenced the computation period.

In summary, considering the complexity of the problem, various topological "patterns" in the SUT models (e.g., density of cycles), principle of individual algorithms, and various

possible criteria $\mathcal{E}$ for determining a winner, formulating the best-performing algorithm that provides the best T is practically impossible for all possible problem instances and combinations of $\mathcal{E}$ and test coverage criteria. However, the current research does not aim to develop an all-encompassing algorithm. In fact, a reasonable and realistic goal is to formulate a strategy to employ multiple best-performing algorithms and produce the best T for the broadest spectrum of G, $\mathcal{E}$, and test coverage criteria. The portfolio strategy presented in this thesis is ideal for achieving this goal.

Regarding the effectiveness of T for detecting complex limited network connectivity defects, the simulated defect node pairs $\chi(G)$ were defined for each SUT model G. For *AllBorderCombinations*, the ANT yielded the best results for the Ψ criterion. The portfolio strategy further outperformed the ANT by 2.7% in Ψ. Limited network connectivity defects of a simple type that can be activated by visiting either the LCZ IN or LCZ OUT node separately would be all activated by the test sets satisfying all test coverage criteria proposed in this thesis. The same situation, including the value of Ψ, occurred with the *EachBorderOnce* test coverage criterion for the defect node pairs.

Considering the *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* criteria, the individual algorithms did not exhibit significant differences between values of Ψ, because the $l(T)$ values of T computed by individual algorithms were relatively similar for *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* criteria (variance of $l(T)$ among all 310 SUT models was 77.92 for *ComprehensiveAllBorderCombinations*, 24.48 for *ComprehensiveEachBorderOnce*, 200.96 for *AllBorderCombinations*, and 80.95 for *EachBorderOnce*). The relative similarity of $l(T)$ for *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* results from the principle of these two test coverage criteria, which demand the edges by which the test case enters an LCZ IN node and by which it leaves an LCZ OUT node to be the Non-LCZ edges.

In addition, the analysis revealed the lower Ψ values for the *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* criteria compared to the *AllBorderCombinations* and *EachBorderOnce* criteria. Although the artificial defects placed in the SUT models might indicate that the "Comprehensive" variants of these test coverage criteria are not effective in terms of the testing expenditure, the *ComprehensiveAllBorderCombinations* and *ComprehensiveEachBorderOnce* criteria are designed for more thorough tests that force test cases to enter and exit the LCZs by Non-LCZ edges, thereby exercising more potential test situations. According to this design, the "Comprehensive" variants of the test coverage criteria are suitable for mission-critical components of the SUT, where relatively higher testing costs are usually acceptable.

For a comparison of T satisfying *AllBorderCombinations*, *EachBorderOnce*, *Compre-*

hensiveAllBorderCombinations, and *ComprehensiveEachBorderOnce* criteria, there are more alternative path-based testing approaches that can be used, not only the *Edge*, *Edge-pair*, and *TDL 3* coverages. For instance, *Prime-path* or even *TDL 4* coverages can be compared. However, the experimental data revealed that T satisfying even *TDL 3* are so large, that further probable increase in $l(T)$ which can be expected for *Prime-path* or *TDL 4* coverage would render this option suitable only as a theoretical baseline.

Chapter 10

Practical Applicability of the Proposed Technique

In this section, we discuss the practical industrial applicability of the proposed LNCT method, primarily designed for current complex IoT systems. However, owing to its suitable level of abstraction, it can be readily applied to various types of systems with electronic and software components connected via a potentially unstable communication network.

In addition to weak, limited, or intermittent data network coverage, situations in which the network signal is jammed can be included in the application of the LNCT method.

Moreover, the presented LNCT technique can be generalized from testing of SUT functionality influenced by limited network connectivity to testing of a system component outage in general. These situations might include testing of situations when a component of the SUT is physically damaged, for instance, in industrial applications, rescue mission management, or defense systems.

Another prospective area involves testing scenarios in which a cyberattack disables the typically vulnerable components of the system. The LNCT can be combined with the current cybersecurity testing techniques [122], [123] to increase the potential effectiveness of these tests.

Other options for LNCT applications include failover testing and testing of disaster-recovery processes, wherein the SUT component can be disabled by internal software errors, data overflow, excessively large user traffic, or general process errors. In one of our papers—*Genetic Algorithm for Path-based Testing of Component Outage Situations in IoT System Processes*—currently under review in IEEE Internet of Things journal, we generalized the problem to test the component under outage.

Generally, the application of the MBT technique implies the necessity of modeling the SUT (in this case, process models) and the probability of limited network connectivity (or

component outage). This modeling requires an initial investment and successive maintenance efforts to update the SUT model with the actual SUT. Thus, the LNCT technique can be advantageous, especially for mission-critical systems in various critical infrastructure domains. In the critical domains, rigorous tests are required to ensure the safety and reliability of the systems. As demonstrated by the experiments, limited network connectivity (or component outage) tests for such systems can be achieved with potentially low effectiveness if only the standard path-based testing methods are employed.

However, LNCT can be applied to non-critical systems as well. The initial investment to SUT modeling and to maintain the model can be returned by the effort saved by automating the generation of test cases. In these cases, the test manager creating a test strategy must assess testing techniques and their costs and benefits. Here, LNCT is a valid option, especially for larger non-critical systems.

Chapter 11

Threats to Validity

Several concerns can be raised regarding the validity of the performed experiments.

11.1 Internal Validity Threats

Firstly, the experimental data might be biased by the nondeterministic nature of SPC, ANT, AGA, and core of TR, the employed set cover algorithm. To mitigate this issue, the computations were executed ten times to obtain the averages of the results along with the worst and best cases for these ten iterations.

Secondly, a question can be raised regarding the TR algorithm that was used as a baseline to compare the discussed algorithms. From available algorithms suitable to serve as the baseline, the proposal by Li *et al.* [13] was the most recent. In addition, a possible bias that might be caused by a wrong implementation of the TR core, the set covering algorithm [13], was prevented by employing the original implementation of the algorithm by Offutt, Ammann, Li, Xu, and Deng[1].

Thirdly, the algorithms were compared using evaluation criteria based on the properties of T (refer to Sections 4.3 and 5.1). The defect detection power, indicating the number of defects in a SUT that would be discovered by a particular T, was analyzed by the simulation of limited network connectivity related defects in the SUT model. The insights into this effectiveness are detailed in Section 8.3. However, additional experiments are required to obtain more accurate data. Typically, to obtain more data, another mutation testing [124], [125] or defect injection experiments [126] are required. However, in principle, limited-network-connectivity defects are difficult to simulate using code mutations or defect insertions. Therefore, an appropriate experiment must be designed to maintain optimal objectivity.

[1] https://cs.gmu.edu/~offutt/softwaretest/coverage-source/

11.2 External Validity Threats

What can be questioned is the definition of the SUT model G. Firstly, whether it shows all the processes that may happen in the system. To mitigate this threat, we don't remove any information from the original process model $\mathcal{G}$ of the SUT during the transformation process into G. Also, there might be doubts whether we have the information about the possible network connectivity issues during a certain process inside SUT present in the model. Therefore, the connection outage probability values are determined to all the transitions in the SUT model G. Lastly, question can be raised whether there is not a need for having parallel edges allowed to be present in G. This threat can be mitigated by replacing each of the parallel edges in G by an auxiliary node with respective edges, which can be than filtered out easily from the test cases in the resulted test set.

The experimental data can be biased because of the small sample size. To mitigate this potential problem, the proposed algorithms were executed in the experiment on all 310 SUT models, which provided sufficiently diverse situations for the algorithms to accurately infer the conclusions. The SUT models are described in Sections 7.2 and 7.4.

Also, concern pertains to the relevance of the SUT models to real-life cases of workflows in existing IoT systems. As the analysis of 50 or more distinct real IoT systems is not feasible because of the confidentiality of the internal structure of typical industrial IoT systems, a balance must be maintained between the number of SUT models used in the experiments and their similarities to real-world cases. The process described in Section 7.2 was applied to ensure that the SUT models used in the experiments are representative to real systems. In the present experiments, we used 163 models created from real IoT projects. Thereafter, we artificially created 119 models inspired by industrial models to resemble their topologies. Ultimately, a dedicated tool was used to artificially generate 28 models for ensuring a sufficient variety of topologies.

Moreover, in path-based testing, the results are strongly influenced by the topology of the SUT model, and this study is no exception. To ensure the best variety in G topology, we employed 310 different SUT models (refer to Table 7.2).

Finally, to compare T satisfying the *EachBorderOnce*, *AllBorderCombinations*, *ComprehensiveEachBorderOnce*, and *ComprehensiveAllBorderCombinations* criteria with an approach that a test engineer would intuitively take to satisfy these given criteria, we used the *Edge*, *Edge-pair*, and *TDL 3* coverage criteria. These criteria were selected because they are the most common and known to the testing community available with and adequate tool support exists for them [12], [34]. Other test coverage criteria such as *Prime path* coverage can be selected for comparison. However, based on the experimental results, the cost of the test cases would be excessively high in such cases. Further experiments are required to explore these cases.

In summary, we believe that the conducted experiments satisfied the established standard practice for experimenting in the MBT field. We base this conclusion on analysis of the common experimental methods reported in relevant MBT publications in field-related impacted journals recognized by the research community (refer to Section 2).

Chapter 12

Conclusions

This thesis proposed a novel technique for the path-based testing of the functionality of an IoT system when the functionality of its components is influenced by limited network connectivity. This technique is based on modeling an SUT process or workflow and determining the network outage probability in the model. Accordingly, the test cases represent the flows of SUT functions such that when the network connectivity is disrupted and restored, and they are sequenced according to a test coverage criterion. In contrast, the number of test steps in other parts of the SUT model is minimal. Based on this principle, the testing costs were minimized in comparison with other ad hoc approaches that can be used to solve the problem. No previous path-based testing algorithms can be directly utilized to this end, because the direct support required to visit a particular node (edge) after another node (edge) is visited in the test case is not included within these algorithms.

The major contributions of this thesis include (1) the formal definition of limited network connectivity testing in an IoT system from the perspective of system functionality, (2) the proposal of four algorithms, including a new SPC based on the principle of the shortest path composition, ANT, which is a novel application of the ACO principle to solve the discussed problem, an AGA algorithm that builds upon the well-known genetic algorithm, and the TR baseline that employs the previous prime-paths generation algorithm by Li *et al.* [13], (3) a detailed evaluation using 310 SUT models, most of which were derived from real-life IoT projects, and (4) the combination of all investigated algorithms in a portfolio strategy to obtain the best T. Considering the defined evaluation criteria (refer to Table 4.1 and Sections 5.1 and 7.3), the ANT algorithm yielded the best results for the majority of the 310 SUT models used in the experiments. However, for certain SUT models, the SPC, AGA, and TR baseline yielded results better than ANT. This effect was further stronger for the "Comprehensive" variants of the defined test coverage criteria (refer to Section 4.2), where only SPC and AGA were comparable alternatives to

ANT algorithm.

We can consider $l(t)$ as the main criterion that approximates the potential effort required to execute test cases to highlight the key findings. Here, we considered the cases in which an algorithm provided clearly better result than the others compared, and if two or more algorithms provided equivalent result, these cases were not counted in statistics. For the *AllBorderCombinations* test coverage criterion, the ANT yielded the best test set for the 140 SUT models, whereas SPC delivered the best set for 6, AGA computed the best set for 45, and TR baseline for 2 models. For *EachBorderOnce*, the ANT clearly computed the best test set for 138 SUT models, SPC for 17, AGA for 57, and the TR baseline for 6 out of 310 models. For *ComprehensiveAllBorderCombinations*, the ANT, SPC, and AGA computed the best test set for 86, 58, and 19 SUT models, respectively. For *ComprehensiveEachBorderOnce*, ANT delivered the best solutions for 109 SUT models, SPC for 33, and AGA for 56.

Overall, two types of limited network-connectivity-related defects were considered in the experiments. A simple type of defect can be activated by separately visiting the LCZ IN or LCZ OUT nodes in the SUT model. This type is activated by any T that satisfies any of the *AllBorderCombinations*, *EachBorderOnce*, *ComprehensiveAllBorderCombinations*, or *ComprehensiveEachBorderOnce* criteria.

More complex defects can be activated by visiting a particular sequence of LCZ IN and LCZ OUT nodes and are simulated by defect node pairs $\chi(G)$; the probability of activating them by T depends on the used algorithm and test coverage criterion.

Here, the potential of the test cases to activate these $\chi(G)$ defects (denoted as Ψ) can be considered the second main criterion regarding the effectiveness of T. For the *AllBorderCombinations* test coverage criterion, ANT yielded the best test set for the 140 SUT models, whereas SPC delivered the best set for 6, AGA produced the best set for 45, and TR baseline for 2 models. For *EachBorderOnce*, ANT computed the best test set for 156 SUT models, SPC for 18, AGA for 47, and the TR baseline for 6 out of 310 models. For *ComprehensiveAllBorderCombinations*, ANT, SPC, and AGA computed the best test set for 86, 58, and 19 SUT models, respectively. For *ComprehensiveEachBorderOnce*, ANT provided the best solution for 110 SUT models, SPC for 57, and AGA for 44. Also, here, we considered only the cases in which an algorithm provided clearly better result than the others compared, and if two or more algorithms provided equivalent result, these situations were not counted in statistics.

Therefore, although ANT renders the best option by the majority of the test set evaluation criteria, all the proposed algorithms must be combined in a portfolio strategy to ensure the best T, which is presented in this thesis. This strategy also provided the best results regarding the potential of the produced test sets to detect simulated limited

connectivity defects.

In future research, the portfolio strategy could be further improved by including another algorithm to compute T. The core principle of this algorithm shall differ primarily from the SPC, ANT, AGA, and TR baseline to increase the probability that the added algorithm will contribute to the computation of a better T for any topology of the SUT model.

A comparison of the proposed portfolio strategy with the test sets satisfying *Edge*, *Edge-pair* and *TDL 3* coverage, selected as the established comparable alternatives, demonstrated that the current proposal evidently outperformed these traditional approaches for the specific case of the limited network connectivity testing problem defined in this thesis.

12.1 Future Directions

We believe that the results of this thesis provide a coherent and effective solution for testing IoT systems operating under limited network connectivity, and this solution can be generalized to testing of the impact of component outages in complex software, IoT, and electronic systems (this capability is further discussed in Section 10).

However, this concept can be further generalized, and during the past two years of research, we identified a novel, more generalized, prospective research stream in path-based testing, which will be explored after completing this PhD project. This stream is **introducing a set of defined constraints as an additional input to an algorithm generating a set of test paths (further $\mathcal{T}$) from the directed-graph-based SUT model $\mathcal{G}$**, all concepts and symbols here now referring to path-based testing preliminaries (Section 2.2).

As such, $\mathcal{T}$ generated from $\mathcal{G}$ by an algorithm must satisfy these constraints.

The constraints can be classified to various types, among which two constraints can be considered elementary building blocks in this case:

1. A function/action/step $n_A \in N \in \mathcal{G}$ must be followed by an action $n_B \in N \in \mathcal{G}$ in a $t \in \mathcal{T}$ and several test steps might be present between actions n_A and n_B. Therefore, the established path-based testing concept of the test requirements (see Section 2.2) is not directly applicable here. Further denoted as *positive constraint*.

2. In analogy, a function/action/step n_A must not be followed by an action n_B in any of $t \in \mathcal{T}$ (this negative condition cannot be handled by the test requirements), denoted as *negative constraint* further.

In addition to these two elementary constraints, we can define more complex constraints.

This thesis practically explores *positive constraints*. In addition, attempts to reflect the constraints in test case generation were made in the DFT discipline discussed in the related work (see Section 2.3). However, the strategies and algorithms in DFT for generating the test paths are proprietarily designed to test the programs at the source code level and are difficult to combine with test coverage criteria suitable for general path-based testing. In any case, previous DFT studies will be further analyzed in more detail, and suitable parts will be utilized in further research.

However, the presence of *negative constraints* is significantly under-researched in path-based testing, similar to the option of constructing a more complex set of constraints.

Regarding the practical applicability of this proposed concept, it will provide path-based testing with new abilities for testing numerous specialized and more complex situations. Starting with limited network connectivity testing and component outage or failover testing, discussed in Section 10, the set of these practical situations can be extended to handle complex test data objects whose wrong states might disrupt path-based testing scenarios or to process exclusive or conditional situations in path-based tests.

Inspired by the established taxonomy and history of the Combinatorial Interaction Testing (CIT) discipline [127], [128], we used the working title **constrained path-based testing** for this concept. This working title follows an analogy from CIT, where Myra Cohen published the first principles of extending the CIT to constrained interaction testing in her PhD thesis in 2004 [129], [130] and influenced the research community to further evolve this sub-discipline (actively participating in this evolution [131], [132]). Currently, constrained interaction testing is a justified, practical, and beneficial substream of CIT, which is routinely used for testing critical systems.

We consider introducing a similar line in path-based testing as an exciting field worth exploring from a research viewpoint and bearing significant potential for industrial practice, and in the upcoming years, we are going to conduct further research in this direction.

Appendix A

List of Publications Related to Thesis Topic

A.1 Published IF Journal Papers

- Matej Klima, Miroslav Bures, Bestoun S. Ahmed, Xavier Bellekens, Robert Atkinson, Christos Tachtatzis and Pavel Herout. Specialized Path-based Technique to Test IoT System Functionality under Limited Network Connectivity. Internet of Things, Elsevier, vol. 22, pp. 100706, 2023. (IF 5.71)

A.2 Published Conference Papers

- Matej Klima, Miroslav Bures, Michaela Kubisova and Pavel Herout. Open Benchmark Testbed to Evaluate Path-based Test Coverage Criteria. Accepted in the 16th IEEE Conference on Software Testing, Verification and Validation (ICST 2023, Core A).

- Matej Klima and Miroslav Bures. A Testing Tool for IoT Systems Operating with Limited Network Connectivity. In Trends and Applications in Information Systems and Technologies (WorldCIST 2021), Advances in Intelligent Systems and Computing, vol. 1367, Springer, pp. 570-576, 2021. DOI: 10.1007/978-3-030-72660-7_54

- Miroslav Bures, Matej Klima, Vaclav Rechtberger, Bestoun S. Ahmed, Hanan Hindy and Xavier Bellekens. Review of Specific Features and Challenges in the Current Internet of Things Systems Impacting Their Security and Reliability. In Trends and Applications in Information Systems and Technologies (WorldCIST 2021), Advances in Intelligent Systems and Computing, vol. 1367, Springer, pp. 546-556, 2021. DOI: 10.1007/978-3-030-72660-7_52

A.3 Registered Patents

- Miroslav Bures and Matej Klima. Method of Testing of IoT System Behaviour in Case of Limited Network Connection. US patent 11,194,700 B2[1]. Holder: CTU in Prague. Registration date: 7.12.2021.
- Miroslav Bures and Matej Klima. Method of Testing of IoT System Behaviour in Case of Limited Network Connection. United Kingdom patent GB2594346[2]. Holder: CTU in Prague. Registration date: 27.10.2021.

This patent is currently being registered also in Germany (2020122113561400DE) and Czechia (PV 2019-804).

A.4 IF Journal Papers Under Review

- Matej Klima, Miroslav Bures, Bestoun S. Ahmed, Hanan Hindy, Xavier Bellekens, and Angelo Gargantini. Genetic Algorithm for Path-based Testing of Component Outage Situations in IoT System Processes. Under review in IEEE Internet of Things (IF 10.24)

[1] https://image-ppubs.uspto.gov/dirsearch-public/print/downloadPdf/11194700
[2] https://www.ipo.gov.uk/p-ipsum/Case/PublicationNumber/GB2594346

Appendix B

List of Other Publications Related to IoT Testing

B.1 Published IF Journal Papers

- Matej Klima, Miroslav Bures, Karel Frajtak, Vaclav Rechtberger, Michal Trnka, Xavier Bellekens, Tomas Cerny and Bestoun S. Ahmed. Selected Code-quality Characteristics and Metrics for IoT Systems. IEEE Access, vol. 10, pp. 46144-46161, 2022. DOI: 10.1109/ACCESS.2022.3170475 (IF 3.48).

- Miroslav Bures, Katerina Neumannova, Pavel Blazek, Matej Klima, Hynek Schvach, Jiri Nema, Michal Kopecky, Jan Dygryn, Vladimir Koblizek. A Sensor Network Utilizing Consumer Wearables for Care of Post-acute COVID-19 Patients. IEEE Internet of Things, vol. 9, issue 23, pp. 23795-23809, 2022. DOI: 10.1109/JIOT.2022.3188914 (IF 10.24)

B.2 Published Conference Papers

- Matej Klima, Vaclav Rechtberger, Miroslav Bures, Xavier Bellekens, Hanan Hindy and Bestoun S. Ahmed. Quality and Reliability Metrics for IoT Systems: A Consolidated View. In 6th EAI International Conference SmartCity 360°, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 372, Springer, 2021. DOI: 10.1007/978-3-030-76063-2_42

- Miroslav Bures, Matej Klima, Vaclav Rechtberger, Xavier Bellekens, Christos Tachtatzis, Robert Atkinson and Bestoun S. Ahmed. Interoperability and Integration Testing Methods for IoT Systems: a Systematic Mapping Study. In SEFM 2020:

Software Engineering and Formal Methods, Lecture Notes in Computer Science, vol. 12310, Springer, pp. 93-112, 2020. DOI: 10.1007/978-3-030-58768-0_6

- Miroslav Bures, Bestoun S. Ahmed, Vaclav Rechtberger, Matej Klima, Michal Trnka, Miroslav Jaros, Xavier Bellekens, Dani Almog and Pavel Herout. PatrIoT: IoT Automated Interoperability and Integration Testing Framework. In 14th IEEE Conference on Software Testing, Verification and Validation (ICST), pp. 454-459, 2021. DOI: 10.1109/ICST49551.2021.00059 (Core A)

B.3 Registered Utility Models

- Miroslav Bures, Vaclav Rechtberger and Matej Klima. Systém pro testování zařízení připojených k internetové síti. Czech Republic Utility Model CZ PUV2019-36985. Holder: CTU in Prague. Registration date: 28.4.2020.

- Miroslav Bures, Vaclav Rechtberger and Matej Klima. System zum Testen von mit dem Internet verbundenen Geräten. Federal Republic of Germany Utility Model DE 20 2020 107 057 U1. Holder: CTU in Prague. Registration date: 10.06.2021.

This utility model is currently being registered also in the Slovak Republic (PP 50077-2020).

B.4 Patent Applications Under Review

- Miroslav Bures, Vaclav Rechtberger and Matej Klima. Postup testování IoT systémů. Czech patent application, 2019, D19132392, PV 2019-802.

Bibliography

[1] S. Li, L. Da Xu, and S. Zhao, "The internet of things: A survey", *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[2] T. Qiu, N. Chen, K. Li, M. Atiquzzaman, and W. Zhao, "How can heterogeneous internet of things build our future: A survey", *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2011–2027, 2018.

[3] A. Čolaković and M. Hadžialić, "Internet of things (iot): A review of enabling technologies, challenges, and open research issues", *Computer Networks*, vol. 144, pp. 17–39, 2018.

[4] A. A. Laghari, K. Wu, R. A. Laghari, M. Ali, and A. A. Khan, "A review and state of art of internet of things (iot)", *Archives of Computational Methods in Engineering*, pp. 1–19, 2021.

[5] B. S. Ahmed, M. Bures, K. Frajtak, and T. Cerny, "Aspects of quality in internet of things (iot) solutions: A systematic mapping study", *IEEE Access*, vol. 7, pp. 13 758–13 780, 2019.

[6] M. Bures, M. Klima, V. Rechtberger, B. S. Ahmed, H. Hindy, and X. Bellekens, "Review of specific features and challenges in the current internet of things systems impacting their security and reliability", in *Trends and Applications in Information Systems and Technologies*, Cham: Springer, 2021, pp. 546–556.

[7] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of things security: A top-down survey", *Computer Networks*, vol. 141, pp. 199–221, 2018.

[8] A. Khanna and S. Kaur, "Internet of things (iot), applications and challenges: A comprehensive review", *Wireless Personal Communications*, vol. 114, pp. 1687–1762, 2020.

[9] B. B. Gupta and M. Quamara, "An overview of internet of things (iot): Architectural aspects, challenges, and protocols", *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, e4946, 2020.

[10] Y. Ding, M. Jin, S. Li, and D. Feng, "Smart logistics based on the internet of things technology: An overview", *International Journal of Logistics Research and Applications*, vol. 24, no. 4, pp. 323–345, 2021.

[11] B. Beizer, *Software testing techniques*. Dreamtech Press, 2003.

[12] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2016.

[13] N. Li, F. Li, and J. Offutt, "Better algorithms to minimize the cost of test paths", in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, IEEE, 2012, pp. 280–289.

[14] M. Klima and M. Bures, "A testing tool for iot systems operating with limited network connectivity", in *Trends and Applications in Information Systems and Technologies*, Springer, 2021, pp. 570–576.

[15] M. Klima, M. Bures, B. S. Ahmed, *et al.*, "Specialized path-based technique to test internet of things system functionality under limited network connectivity", *Internet of Things*, vol. 22, p. 100 706, 2023.

[16] M. Bures, K. Neumannova, P. Blazek, *et al.*, "A sensor network utilizing consumer wearables for telerehabilitation of post-acute covid-19 patients", *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 795–23 809, 2022. DOI: 10.1109/JIOT.2022.3188914.

[17] A. Pretschner, "Model-based testing", in *Proceedings. 27th International Conference on Software Engineering (ICSE)*, 2005, pp. 722–723. DOI: 10.1109/ICSE.2005.1553582.

[18] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: A systematic review", in *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, ser. WEASELTech '07, New York, NY, USA: Association for Computing Machinery, 2007, 31–36. DOI: 10.1145/1353673.1353681.

[19] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches", *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012. DOI: 10.1002/stvr.456.

[20] S. Dalal, A. Jain, N. Karunanithi, *et al.*, "Model-based testing in practice", in *Proceedings of the 1999 International Conference on Software Engineering*, 1999, pp. 285–294. DOI: 10.1145/302405.302640.

[21] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "Mobiguitar: Automated model-based testing of mobile apps", *IEEE Software*, vol. 32, no. 5, pp. 53–59, 2015. DOI: 10.1109/MS.2014.55.

[22] L. Villalobos, C. Quesada-López, A. Martinez, and M. Jenkins, "Model-based testing areas, tools and challenges: A tertiary study", *CLEI Electronic Journal*, vol. 22, Apr. 2019. DOI: 10.19153/cleiej.22.1.3.

[23] E. Bringmann and A. Krämer, "Model-based testing of automotive systems", in *2008 1st International Conference on Software Testing, Verification, and Validation*, 2008, pp. 485–493. DOI: 10.1109/ICST.2008.45.

[24] J. Chang, D. J. Richardson, and S. Sankar, "Structural specification-based testing with adl", in *International Symposium on Software testing and analysis (ISSTA) '96*, 1996.

[25] A. Abdurazik and J. Offutt, "Using uml collaboration diagrams for static checking and test generation", in *Lecture Notes in Computer Science*, vol. 1939, Jan. 2001. DOI: 10.1007/3-540-40011-7_28.

[26] Y. Wu, M.-H. Chen, and J. Offutt, "Uml-based integration testing for component-based software", in *Lecture Notes in Computer Science*, vol. 2580, Feb. 2003, pp. 251–260. DOI: 10.1007/3-540-36465-X_24.

[27] P. K. Arora and R. Bhatia, "A systematic review of agent-based test case generation for regression testing", *Arabian Journal for Science and Engineering*, vol. 43, no. 2, pp. 447–470, Feb. 2018. DOI: 10.1007/s13369-017-2796-4.

[28] T. Su, K. Wu, W. Miao, *et al.*, "A survey on data-flow testing", *ACM Comput. Surv.*, vol. 50, no. 1, 2017. DOI: 10.1145/3020266.

[29] M. Shirole and R. Kumar, "Uml behavioral model based test case generation: A survey", *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–13, 2013.

[30] V. Arora, R. Bhatia, and M. Singh, "Synthesizing test scenarios in uml activity diagram using a bio-inspired approach", *Computer Languages, Systems & Structures*, vol. 50, pp. 1–19, 2017.

[31] M. Bures and B. S. Ahmed, "Employment of multiple algorithms for optimal path-based test selection strategy", *Information and Software Technology*, vol. 114, pp. 21–36, 2019.

[32] M. Bures, B. S. Ahmed, and K. Z. Zamli, "Prioritized process test: An alternative to current process testing strategies", *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 07, pp. 997–1028, 2019.

[33] S. Anand, E. K. Burke, T. Y. Chen, *et al.*, "An orchestrated survey of methodologies for automated software test case generation", *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[34] M. Vroon, B. Broekman, T. Koomen, and L. van der Aalst, *TMap next: for result-driven testing*. Uitgeverij kleine Uil, 2013.

[35] N. Li, "Generating cost-effective criteria-based tests from behavioral models", Ph.D. dissertation, George Mason University, 2014.

[36] M. Bures and V. Rechtberger, "Dynamic data consistency tests using a crud matrix as an underlying model", in *Proceedings of the 2020 European Symposium on Software Engineering*, 2020, pp. 72–79.

[37] P. Herman, "A data flow analysis approach to program testing", *Australian Computer Journal*, vol. 8, no. 3, pp. 92–96, 1976.

[38] J. Badlaney, R. Ghatol, and R. Jadhwani, "An introduction to data-flow testing", North Carolina State University, Dept. of Computer Science, 2006.

[39] S. Rapps and E. J. Weyuker, "Data flow analysis techniques for test data selection", in *Proceedings of the 6th International Conference on Software Engineering*, ser. ICSE '82, Washington, DC, USA: IEEE Computer Society Press, 1982, 272–278.

[40] A. Khamis, R. Bahgat, and R. Abdelaziz, "Automatic test data generation using data flow information", vol. 2, Doğuş Üniversitesi Dergisi, 2000, pp. 140–153.

[41] B. Korel, "Automated software test data generation", *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 870–879, 1990. DOI: 10.1109/32.57624.

[42] F. E. Allen, "Control flow analysis", in *Proceedings of a Symposium on Compiler Optimization*, New York, NY, USA: Association for Computing Machinery, 1970, 1–19. DOI: 10.1145/800028.808479.

[43] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial", *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*, pp. 1–59, 2012.

[44] E. H. Houssein, M. Younan, and A. E. Hassanien, "Nature-inspired algorithms: A comprehensive review", *Hybrid Computational Intelligence*, pp. 1–25, 2019.

[45] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications", King's College London, Tech. Rep. TR-09-03, May 2009.

[46] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing", in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–12. DOI: 10.1109/ICST.2015.7102580.

[47] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering", *Automated Software Engineering*, vol. 24, no. 2, pp. 233–294, Jun. 2017. DOI: 10.1007/s10515-017-0215-4.

[48] L. Briand and Y. Labiche, "A uml-based approach to system testing", *Software and Systems Modeling*, vol. 1, no. 1, pp. 10–42, Sep. 2002. DOI: 10.1007/s10270-002-0004-8.

[49] M. Harman, "The current state and future of search based software engineering", in *Future of Software Engineering (FOSE '07)*, 2007, pp. 342–357. DOI: 10.1109/FOSE.2007.29.

[50] P. R. Srivastava, N. Jose, S. Barade, and D. Ghosh, "Optimized test sequence generation from usage models using ant colony optimization", *International Journal of Software Engineering &Applications*, vol. 2, no. 2, pp. 14–28, 2010.

[51] S. S. B. Lam, M. H. P. Raju, S. Ch, P. R. Srivastav, *et al.*, "Automated generation of independent paths and test suite optimization using artificial bee colony", *Procedia Engineering*, vol. 30, pp. 191–200, 2012.

[52] A. Layeb and Z. Benayad, "A novel firefly algorithm based ant colony optimization for solving combinatorial optimization problems", *International Journal of Computer Science and Applications*, vol. 11, p. 19, Dec. 2014.

[53] A. Windisch, S. Wappler, and J. Wegener, "Applying particle swarm optimization to software testing", in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '07, New York, NY, USA: Association for Computing Machinery, 2007, 1121–1128. DOI: 10.1145/1276958.1277178.

[54] *Applied Nature-Inspired Computing: Algorithms and Case Studies*. Springer Singapore, 2020. DOI: 10.1007/978-981-13-9263-4.

[55] I. Hermadi and M. Ahmed, "Genetic algorithm based test data generator", in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 1, 2003, 85–91 Vol.1. DOI: 10.1109/CEC.2003.1299560.

[56] I. Hermadi, C. Lokan, and R. Sarker, "Genetic algorithm based path testing: Challenges and key parameters", in *2010 Second World Congress on Software Engineering*, vol. 2, 2010, pp. 241–244. DOI: 10.1109/WCSE.2010.82.

[57] J. Wegener, A. Baresel, and H. Sthamer, "Suitability of evolutionary algorithms for evolutionary testing", in *Proceedings 26th Annual International Computer Software and Applications*, 2002, pp. 287–289. DOI: 10.1109/CMPSAC.2002.1044566.

[58] N. Mansour and M. Salame, "Data generation for path testing", *Software Quality Journal*, vol. 12, no. 2, pp. 121–136, Jun. 2004. DOI: 10.1023/B:SQJO.0000024059.72478.4e.

[59] W. Xibo and S. Na, "Automatic test data generation for path testing using genetic algorithms", in *2011 Third International Conference on Measuring Technology and Mechatronics Automation*, vol. 1, 2011, pp. 596–599. DOI: 10.1109/ICMTMA.2011.152.

[60] M. Dorigo, "Optimization, learning and natural algorithms", Ph.D. dissertation, Politecnico di Milano, 1992.

[61] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[62] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization", *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006. DOI: 10.1109/MCI.2006.329691.

[63] P. R. Srivastava, K. Baby, and G Raghurama, "An approach of optimal path generation using ant colony optimization", in *TENCON 2009-2009 IEEE Region 10 Conference*, IEEE, 2009, pp. 1–6.

[64] F. Sayyari and S. Emadi, "Automated generation of software testing path based on ant colony", in *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*, IEEE, 2015, pp. 435–440.

[65] A. S. Ghiduk, "A new software data-flow testing approach via ant colony algorithms", *Universal Journal of Computer science and engineering Technology*, vol. 1, no. 1, pp. 64–72, 2010.

[66] J. H. Holland *et al.*, "Adaptation in natural and artificial systems", *University of Michigan Press*, 1975.

[67] L. Davis, "Handbook of genetic algorithms", *Cumulative Index about publications in Computer Aided Architectural Design (CumInCAD)*, 1991.

[68] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986. DOI: 10.1109/TSMC.1986.289288.

[69] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st. Addison-Wesley Longman Publishing Co., Inc., 1989.

[70] B. F. Jones, D. E. Eyres, and H.-H. Sthamer, "A strategy for using genetic algorithms to automate branch and fault-based testing", *The Computer Journal*, vol. 41, no. 2, pp. 98–107, Jan. 1998. DOI: 10.1093/comjnl/41.2.98.

[71] R. P. Pargas, M. J. Harrold, and R. R. Peck, "Test-data generation using genetic algorithms", *Software Testing, Verification and Reliability*, vol. 9, no. 4, pp. 263–282, 1999. DOI: 10.1002/(SICI)1099-1689(199912)9:4<263::AID-STVR190>3.0.CO;2-Y.

[72] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing", *Information and Software Technology*, vol. 43, no. 14, pp. 841–854, 2001. DOI: 10.1016/S0950-5849(01)00190-2.

[73] D. P. Srivastava and T.-H. Kim, "Application of genetic algorithm in software testing", *International Journal of Software Engineering and Its Applications*, vol. 3, Nov. 2009.

[74] A. S. Ghiduk, "Automatic generation of basis test paths using variable length genetic algorithm", *Information Processing Letters*, vol. 114, no. 6, pp. 304–316, 2014.

[75] M. Girgis, A. Ghiduk, and E. Abd-Elkawy, "Automatic generation of data flow test paths using a genetic algorithm", *International Journal of Computer Applications*, vol. 89, 2014. DOI: 10.5120/15684-4534.

[76] B. Hoseini and S. Jalili, "Automatic test path generation from sequence diagram using genetic algorithm", in *7'th International Symposium on Telecommunications (IST'2014)*, IEEE, 2014, pp. 106–111.

[77] J.-C. Lin and P.-L. Yeh, "Using genetic algorithms for test case generation in path testing", in *Proceedings of the Ninth Asian Test Symposium*, 2000, pp. 241–246. DOI: 10.1109/ATS.2000.893632.

[78] R. Khan, M. Amjad, and A. K. Srivastava, "Optimization of automatic generated test cases for path testing using genetic algorithm", in *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT)*, IEEE, 2016, pp. 32–36.

[79] X. Bao, Z. Xiong, N. Zhang, J. Qian, B. Wu, and W. Zhang, "Path-oriented test cases generation based adaptive genetic algorithm", *PloS one*, vol. 12, no. 11, e0187471, 2017.

[80] C. Sharma, S. Sabharwal, and R. Sibal, "Applying genetic algorithm for prioritization of test case scenarios derived from UML diagrams", *CoRR*, 2014. arXiv: 1410.4838. [Online]. Available: http://arxiv.org/abs/1410.4838.

[81] M. R. Girgis, "An experimental evaluation of a symbolic execution system", *Software Engineering Journal*, vol. 7, no. 4, 285–290, 1992. DOI: 10.1049/sej.1992.0029.

[82] D. Karaboga, "An idea based on honey bee swarm for numerical optimization", Erciyes University, Tech. Rep. TR06, 2005.

[83] X.-S. Yang, "Firefly algorithms for multimodal optimization", in *Stochastic Algorithms: Foundations and Applications*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 169–178.

[84] P. R. Srivatsava, B Mallikarjun, and X.-S. Yang, "Optimal test sequence generation using firefly algorithm", *Swarm and Evolutionary Computation*, vol. 8, pp. 44–53, 2013.

[85] T.-B. Tan and W.-K. Cheng, "Software testing levels in internet of things (iot) architecture", in *New Trends in Computer Technologies and Applications*, Singapore: Springer Singapore, 2019, pp. 385–390.

[86] G. Murad, A. Badarneh, A. Qusef, and F. Almasalha, "Software testing techniques in iot", in *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, 2018, pp. 17–21. DOI: 10.1109/CSIT.2018.8486149.

[87] J. P. Dias, F. Couto, A. C. Paiva, and H. S. Ferreira, "A brief overview of existing tools for testing the internet-of-things", in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, pp. 104–109. DOI: 10.1109/ICSTW.2018.00035.

[88] S. Muthiah and R. Venkatasubramanian, "The internet of things: Qa unleashed", in *Cognizant 20-20 Insights*, Cognizant, 2015.

[89] M. Sirshar, K. Naeem, M. Khan, and T. Akbar, "Software quality assurance testing methodologies in iot", Preprints.org, 2019. (visited on 02/02/2023).

[90] J. Esquiagola, L. C. de Paula Costa, P. Calcina, G. Fedrecheski, and M. Zuffo, "Performance testing of an internet of things platform.", in *IoTBDS*, 2017, pp. 309–314.

[91] H. Rudeš, I. N. Kosović, T. Perković, and M. Čagalj, "Towards reliable iot: Testing lora communication", in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, IEEE, 2018, pp. 1–3.

[92] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in iot: A systematic mapping", *Journal of Systems and Software*, vol. 132, pp. 186–203, 2017.

[93] T. Kanstrén, J. Mäkelä, and P. Karhula, "Architectures and experiences in testing iot communications", in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, pp. 98–103. DOI: 10.1109/ICSTW.2018.00034.

[94] A. P. Matz, J.-A. Fernandez-Prieto, U. Birkel, *et al.*, "A systematic analysis of narrowband iot quality of service", *Sensors*, vol. 20, no. 6, p. 1636, 2020.

[95] M. Singh and G. Baranwal, "Quality of service (qos) in internet of things", in *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2018, pp. 1–6. DOI: 10.1109/IoT-SIU.2018.8519862.

[96] H. Kim, A. Ahmad, J. Hwang, *et al.*, "Iot-taas: Towards a prospective iot testing framework", *IEEE Access*, vol. 6, pp. 15 480–15 493, 2018. DOI: 10.1109/ACCESS.2018.2802489.

[97] A. K. Gomez and S. Bajaj, "Challenges of testing complex internet of things (iot) devices and systems", in *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, 2019, pp. 1–4. DOI: 10.1109/KSE.2019.8919324.

[98] B. Crews and S. Mangal. "Iot and it's impact on testing". (2019), [Online]. Available: https://smartbear.com/blog/internet-of-things-101/ (visited on 03/01/2023).

[99] M. Bures, T. Cerny, and B. S. Ahmed, "Internet of things: Current challenges in the quality assurance and testing methods", in *International Conference on Information Science and Applications*, Springer, 2018, pp. 625–634.

[100] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges", *Mobile Networks and Applications*, vol. 24, no. 3, pp. 796–809, 2018. DOI: 10.1007/s11036-018-1089-9.

[101] M. Bures, T. Cerny, and M. Klima, "Prioritized process test: More efficiency in testing of business processes and workflows", in *International Conference on Information Science and Applications*, Springer, 2017, pp. 585–593.

[102] A. Dwarakanath and A. Jankiti, "Minimum number of test paths for prime path and other structural coverage criteria", in *IFIP International Conference on Testing Software and Systems*, Springer, 2014, pp. 63–79.

[103] M. Dorigo and G. Di Caro, "Ant colony optimization: A new meta-heuristic", in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 2, 1999, 1470–1477 Vol. 2. DOI: 10.1109/CEC.1999.782657.

[104] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future", *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021. DOI: 10.1007/s11042-020-10139-6.

[105] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals", *University computing*, vol. 15, no. 2, pp. 56–69, 1993.

[106] M. Girgis, "Automatic test data generation for data flow testing using a genetic algorithm.", *Journal of Universal Computer Science*, vol. 11, pp. 898–915, Jan. 2005.

[107] K. Jebari and M. Madiafi, "Selection methods for genetic algorithms", *International Journal of Emerging Sciences*, vol. 3, no. 4, pp. 333–344, 2013.

[108] Y. Suresh and S. K. Rath, "A genetic algorithm based approach for test data generation in basis path testing", *CoRR*, vol. abs/1401.5165, 2014. arXiv: 1401.5165. [Online]. Available: http://arxiv.org/abs/1401.5165.

[109] H. Chiroma, S. Abdulkareem, A. Abubakar, A. Zeki, A. Y. Gital, and M. J. Usman, "Correlation study of genetic algorithm operators: Crossover and mutation probabilities", in *Proceedings of the International Symposium on Mathematical Sciences and Computing Research*, 2013, pp. 6–7.

[110] M. Bures, "Pctgen: Automated generation of test cases for application workflows", in *New Contributions in Information Systems and Technologies*, Springer, 2015, pp. 789–794.

[111] V. Aravindan and D. James, "Smart homes using internet of things", *International Research Journal of Engineering and Technology*, pp. 1725–1729, 2017.

[112] S. Memusi and C. Kpotosu, "Smart agriculture using iot", May 2018. DOI: 10.13140/RG.2.2.17522.86726.

[113] M. S. Hossain, M. Rahman, M. T. Sarker, M. E. Haque, and A. Jahid, "A smart iot based system for monitoring and controlling the sub-station equipment", *Internet of Things*, vol. 7, p. 100 085, 2019. DOI: 10.1016/j.iot.2019.100085.

[114] A. K. Gupta and R. Johari, "Iot based electrical device surveillance and control system", in *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2019, pp. 1–5. DOI: 10.1109/IoT-SIU.2019.8777342.

[115] A. Muneer, S. Fati, and S. Fuddah, "Smart health monitoring system using iot based smart fitness mirror", *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 18, pp. 317–331, Feb. 2020. DOI: 10.12928/TELKOMNIKA.v18i1.12434.

[116] G. K. Jakir Hussain, O Dharshini, G Kavipriya, and G Jeevanandhini, "E-parking reservation system based on iot for smart cities", *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 8, pp. 1981–1986, Mar. 2019. DOI: 10.15680/IJIRSET.2019.0802044.

[117] G. Susrama, N. R.G, S. Winardi, *et al.*, "Progressive parking smart system in surabaya's open area based on iot", *Journal of Physics Conference Series*, vol. 1569, p. 022 043, Jul. 2020. DOI: 10.1088/1742-6596/1569/2/022043.

[118] R. P. Ch, "Patient health monitoring using iot", *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, pp. 454–457, Dec. 2018.

[119] H. Fuentes and D. Mauricio, "Smart water consumption measurement system for houses using iot and cloud computing", *Environmental Monitoring and Assessment*, vol. 192, no. 9, p. 602, 2020. DOI: 10.1007/s10661-020-08535-4.

[120] M. Hadipour, J. F. Derakhshandeh, M. A. Shiran, and R. Rezaei, "Automatic washing system of led street lighting via internet of things", *Internet of Things*, vol. 1-2, pp. 74–80, 2018. DOI: 10.1016/j.iot.2018.08.006.

[121] E. Priyanka, C. Maheswari, and S. Thangavel, "Iot based field parameters monitoring and control in press shop assembly", *Internet of Things*, vol. 3-4, pp. 1–11, 2018. DOI: 10.1016/j.iot.2018.09.004.

[122] R. Leszczyna, "Review of cybersecurity assessment methods: Applicability perspective", *Computers & Security*, vol. 108, p. 102 376, 2021.

[123] F. Luo, X. Zhang, Z. Yang, *et al.*, "Cybersecurity testing for automotive domain: A survey", *Sensors*, vol. 22, no. 23, p. 9211, 2022.

[124] P. Reales, M. Polo, J. L. Fernandez-Aleman, A. Toval, and M. Piattini, "Mutation testing", *IEEE software*, vol. 31, no. 3, pp. 30–35, 2014.

[125] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing", *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[126] M. Bures, P. Herout, and B. S. Ahmed, "Open-source defect injection benchmark testbed for the evaluation of testing", in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, IEEE, 2020, pp. 442–447.

[127] D. R. Kuhn, R. Bryce, F. Duan, L. S. Ghandehari, Y. Lei, and R. N. Kacker, "Combinatorial testing: Theory and practice", *Advances in computers*, vol. 99, pp. 1–66, 2015.

[128] R. Tzoref-Brill, "Advances in combinatorial testing", *Advances in Computers*, vol. 112, pp. 79–134, 2019.

[129] M. B. Cohen, "Designing test suites for software interactions testing", Ph.D. dissertation, The University of Auckland, 2004.

[130] B. S. Ahmed, K. Z. Zamli, W. Afzal, and M. Bures, "Constrained interaction testing: A systematic literature study", *IEEE Access*, vol. 5, pp. 25 706–25 730, 2017.

[131] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "Evaluating improvements to a meta-heuristic search for constrained interaction testing", *Empirical Software Engineering*, vol. 16, pp. 61–102, 2011.

[132] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach", *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, 2008.