Internet of Things 22 (2023) 100706

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Specialized path-based technique to test Internet of Things system functionality under limited network connectivity

Matej Klima[a], Miroslav Bures[a,*], Bestoun S. Ahmed[b,a], Xavier Bellekens[c], Robert Atkinson[d], Christos Tachtatzis[d], Pavel Herout[e]

[a] *System Testing IntelLigent Lab (STILL), Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo namesti 13, Prague, 121 35, Czechia*
[b] *Department of Mathematics and Computer Science, Karlstad University, Universitetsgatan 2, Karlstad, 651 88, Sweden*
[c] *Lupovis.io, 204 George Street,, Glasgow, G1 1XW, Scotland, UK*
[d] *Department of Electronic and Electrical Engineering, University of Strathclyde, 204 George Street,, Glasgow, G1 1XW, Scotland, UK*
[e] *Department of Computer Science and Engineering, University of West Bohemia, Univerzitni 2732/8, Pilsen, 301 00, Czechia*

ARTICLE INFO

Keywords:
Internet of Things
Reliability
Path-based testing
Model-based testing
Test automation
Limited network connectivity
Test case generation

ABSTRACT

Contemporary Internet-of-Things (IoT) systems are hindered by several reliability-related issues, especially, the dynamic behavior of IoT systems caused by limited and often unstable network connectivity. Several intuitive ad-hoc approaches can be employed to test this behavior; however, the effectiveness of these approaches in detecting defects and their overall testing costs remain questionable. Therefore, we present a new specialized path-based technique to test the processes of an IoT system in scenarios wherein parts of these processes are influenced by limited or disrupted network connectivity. The proposed technique can be scaled using two levels of test coverage criteria to determine the strengths of the test cases. For this purpose, we propose two algorithms for generating test cases to implement the technique: an ant colony optimization-based search and a graph-traversal-based test case composition. We compared the efficiency of the proposed approach with possible solutions obtained using a standard path-based testing approach based on prime paths computed by a set-covering algorithm. We consider the total number of test case steps as the main proxy for test effort in experiments employing 150 problem models. For the less intensive of the two used test-coverage criteria, EachBorderOnce, an ant colony optimization-based algorithm, produced test sets with the same averaged number of steps as the graph traversal-based test-case composition; however, this algorithm performed with averaged number of steps 10% lower than a prime paths-based algorithm. For the more intensive test coverage criterion, AllBorderCombinations, these differences favoring the ant colony optimization-based algorithm were 18% and 25%, respectively. For these two types of defined test coverage criteria, the ant colony optimization-based search, graph-traversal-based algorithm, and standard path-based testing approach based on prime paths achieved the best results for 93 and 78, 14 and 24, and 13 and 17 models for AllBorderCombinations and EachBorderOnce criterion, respectively. Therefore, to guarantee the best test set, all compared algorithms are combined in a portfolio strategy that yields the best results based on the potential of the produced test sets to detect simulated defects caused by

* Corresponding author.
E-mail address: miroslav.bures@fel.cvut.cz (M. Bures).

https://doi.org/10.1016/j.iot.2023.100706
Received 13 October 2022; Received in revised form 18 December 2022; Accepted 24 January 2023
Available online 3 February 2023

limited network connectivity. Additionally, this portfolio strategy also yields test sets, implying the lowest test effort for experimental problem instances.

1. Introduction

Internet-of-Things (IoT) systems have progressed significantly over the last decade; these systems have evolved from an initial hype to everyday technological realities that are integrated into people's work processes and lives [1,2]. This growth in IoT systems has introduced new significant challenges in terms of quality, usability, security, and reliability [3–5]. In this regard, ensuring the reliable behavior of a dynamic IoT system when parts of the system or its processes need to operate with limited or unstable network connectivity [3,4] is a critical issue. In this study, limited connectivity refers to a complete network connectivity outage, intermittent connectivity, significantly low bandwidth, high network-error rate, or any state that negatively influences the reliability and functionality of an IoT system.

Generally, the server side (back-end) is spatially static and connected to a stable network in a web-based client–server architecture for information systems. In contrast, the client side can move physically and are subject to limited network connectivity or temporary network outages. This scenario is typically observed in rural or maritime areas with weak or no wireless network coverage or for tunnels in urban areas. In such applications, users are prepared to endure network connectivity issues and interact with the system accordingly. However, in the case of dynamic IoT systems, connected devices, such as sensors, actuators, or back-end infrastructure, can move spatially. Therefore, they are relatively more sensitive to limited or disrupted network connectivity. Certain types of IoT systems are affected by limited or disrupted network connectivity. Examples of dynamic sensor networks wherein the geographical location of devices changes during system operation include smart farming, intelligent transportation, smart cars, and military and logistic systems [6,7].

The reliability of a service provided by a system to its users must be maintained if the network connectivity of particular parts of an IoT system is limited or disrupted. In such a scenario, the functionality of the system should not be disrupted by an explicit defect caused by a network-connectivity outage. In such a case, users can accept the restricted system functionality; however, they must be notified in adequate time. The system cannot interrupt actual transactions, lose data, revert to an unexpected state, crash, or become unresponsive. In dynamic IoT systems subjected to network connectivity limitations, system behavior must be tested under these conditions. Moreover, testing must be performed optimally to ensure the effective detection of relevant defects and reduce the costs associated with such testing [3,8], which is the main motivation driving our proposal.

Currently, established path-based testing techniques [9,10] can be applied only partially to test a System Under Test (SUT) under limited network connectivity. By analyzing all the path-based testing and data-flow algorithms mentioned in this paper, could not identify any algorithm that directly addressed the problem described in Section 3 and formally defined further in Section 4.

As we further explain in Section 2.1, the only type of algorithms that can be utilized in this scenario are those that accept the SUT model $\mathcal{G}$ or its variant based on a directed graph and a set of test requirements. Certain examples are reported by Li et al. [9] for prime paths, namely brute force, prefix-graph-based, and set-covering algorithms. In our proposal, we utilized the set-covering algorithm and augmented with a special procedure to prepare the test requirements, as presented in Section 7.3. This algorithm serves as a baseline for evaluating original algorithms designed directly to solve the limited network connectivity problem, presented in this paper.

Therefore, this study proposes a limited network connectivity test (LNCT) technique, which is a new and specialized technique for generating test cases to test the behavior of an IoT system under limited network connectivity. The contributions of this study are as follows:

1. Three algorithms are presented to generate test cases for the path-based testing approach to limited connectivity testing in IoT systems. These algorithms include two novel algorithms designed specifically for the discussed problem in a way that has not been employed in the previous literature. These two algorithms are based on the shortest-path composition principle and a novel application of the ACO principle. The third algorithm is the baseline formulated by extending a previous algorithm that uses the established concept of test requirements.
2. An approach based on the concept of the test requirements is compared with specific algorithms designed for the discussed case.
3. An evaluation study is conducted with experimental data wherein the proprieties of the generated test cases are discussed comparatively. The cost of testing measured in the number of test steps and the potential of test cases to detect defects present in an SUT are investigated.
4. A portfolio strategy employing all related algorithms is presented to achieve the optimal result for various possible SUT models in industrial settings, which is an approach not previously reported in the literature for this specific type of problem.

The remainder of this paper is organized as follows. Section 2 analyzes the related work and motivation behind the proposed technique. Section 3 presents the principle of the proposed technique, and Section 4 defines the SUT model utilized in the technique. Sections 5 and 6 present the test coverage and evaluation criteria, respectively. Section 7 presents the three algorithms that generate test sets from the SUT model. Section 8 describes the experimental design and presents the results of individual algorithms. Section 9 details a portfolio strategy combining the algorithms and summarizes the results. Section 10 discusses the experimental results. Section 11 analyzes the threats to validity and presents steps for minimizing their possible effects. Finally, in the last section, conclusions are drawn based on the findings of this study.

2. Related work

There are three primary directions of related work that must be analyzed considering our proposal: (1) existing path-based testing techniques and algorithms, (2) existing data-flow testing techniques that overlap with path-based testing techniques, (3) ant colony optimization and nature-inspired algorithms, and (4) alternative approaches for reliability testing of IoT systems under conditions of weak network coverage.

2.1. Path-based testing

Starting with the state-of-the-art path-based testing, an established model of an SUT is available in the literature, and a number of algorithms that generate test cases for various coverage criteria have been published [9,11–14]. The typical notation of an SUT model in path-based testing is based on a directed graph $\mathcal{G} = (N, E, n_s, N_e)$, where N is a nonempty finite set of nodes, $E \subseteq N \times N$ is a finite set of edges, n_s denotes the 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 [9,10,12]. The nodes model decision points, actions, or function calls in an SUT workflow or process, and the edges model transitions between them.

A test case is defined as the path from n_s to any node from N_e. The set of test cases must satisfy a defined test-coverage criterion. The edge, edge pair, and prime path coverage are the most common criteria [10,12].

Individual algorithms differ in their ability to provide a near-to-optimum solution satisfying the given test coverage criteria [9,11–14]. Therefore, combining them with a portfolio strategy is a practical alternative from the perspective of a testing practitioner [12]. However, these common algorithms fail to address the goal of our study directly because they are designed for common path-based testing problems, which are different from the specific goal of limited network connectivity testing.

The reasons for this are as follows: Considering the goal and principle of the proposed technique, as explained in Section 3, the existing concepts that allow the touring of the defined sequences of two non-adjacent edges in a path need to be analyzed; here, the main established concept is a *set of test requirements*. The test requirements are paths in G that must be presented as subpaths in the test cases generated for the SUT model [10].

If not explicitly designed to satisfy a particular test-coverage criterion, algorithms that create test cases usually accept test requirements as an input, such as those reported by Li et al. [9]. The test requirements are a partially applicable concept for solving defined problems.

We define $R = \{r_{interrupted}, r_{restored}\}$ to model our problem using test requirements, and assume only one subprocess exists for the tested process (a subgraph of $\mathcal{G}$) wherein the network connection can be interrupted and restored. Herein, $r_{interrupted}$ and $r_{restored}$ are one edge long; $r_{interrupted}$ models an SUT function wherein the network connectivity is interrupted, and $r_{restored}$ models an SUT function therein the network connectivity is restored. In this example, $r_{interrupted}$ must be followed by $r_{restored}$ in the test case, and the path from n_s to any node from N_e must be minimal.

However, algorithms that accept R as the input optimize the final set of test cases for their minimal length, and no factor guarantees that $r_{interrupted}$ will be followed by $r_{restored}$. To the best of our knowledge, no algorithm accepts additional constraints that define the required order of the input test requirements. Furthermore, no motivation exists for achieving such functionality in standard path-based testing, and the algorithm that generates such a test set becomes unnecessarily complex. However, in our problem described in Section 3, such functionality is essential. Thus, our proposal is an original contribution to the field of limited network connectivity testing.

The concept of test requirements can be partially utilized when, in the discussed example, $R = \{\text{a path from } r_{interrupted} \text{ to } r_{restored}\}$. Certain algorithms that accept R can be employed [9] and must be accompanied by an additional algorithm that can prepare a set of test requirements. Furthermore, in this study, we employed such an algorithm, EPP, as a baseline for the experimental evaluation of the proposed algorithms.

It cannot be validated if such an approach would guarantee the best solution because a greater number of pairs of "$r_{interrupted}$ and $r_{restored}$" may exist in the model. This aspect motivated us to explore new alternative algorithms, aiming to derive an optimal solution to the limited network connectivity testing problem, as considered in this study.

2.2. Data-flow testing

Another field that may provide relevant work is that of data-flow testing (DFT) [15], which overlaps with general path-based testing. Dynamic DFT is relevant to this study based on the available DFT subtypes. As the main working principle driving DFT, the SUT variables are verified by inspecting their definitions and uses, which is achieved by extracting *definition-use (def-use) pairs* from the code. Each pair is examined with respect to the selected coverage criteria [15]. Although several coverage criteria are available for utilization, research suggests that the all-uses criterion, which requires covering every definition at least once and using an association in the program, is the most effective approach for detecting defects [16]. The dynamic data-flow testing process includes (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.

Generally, the construction of path-based test cases that contain defined def-use pairs is a potentially applicable concept for solving the problem of limited network connectivity testing as discussed in this study. However, DFT techniques potentially fail to solve the problem entirely owing to the following reasons presented in this section.

Although the CFG (constructed from the source code of the SUT) is used as the underlying model of the problem, it differs from our problem model specified in Section 4, rendering this problem definition and solution an original contribution to testing research and praxis. The idea of determining paths containing specific pairs of nodes in the CFG and subsequently ensuring their presence in the test cases is similar to sequencing $r_{interrupted}$ and $r_{restored}$ SUT-model elements.

However, for the coverage criteria specified in Section 5, it is important to restrict the path between $r_{interrupted}$ and $r_{restored}$ to lead inside the SUT model part affected by limited network connectivity. Specifically, the path should not leave it from any other node than those present in $r_{restored}$. This requirement can be theoretically satisfied by the definition of adequate def-use pairs in an SUT model. An algorithm, defining these def-use pairs to satisfy the test coverage criteria defined in our proposal, must be formulated as the first stage of the computation. However, such a solution is unlikely to yield near-optimum results from the viewpoints of complexity and path construction principle. The test paths should be kept minimal while satisfying the test coverage criteria, which is the reason underlying the aforementioned inadequacy. Hence, the definition of a completely new algorithm, which is designed directly to solve the defined problem, renders a more viable solution with a higher probability of yielding near-optimum results.

2.3. *Ant colony optimization and nature inspired algorithms*

A number of algorithms for solving combinatorial problems have been inspired by nature. In addition to genetic algorithms, a pioneering algorithm is the Particle Swarm Optimization algorithm (PSO), which is for instance employed in communication protocols [17] as well as in system testing [18,19]. In PSO, search space exploration imitates the behavior of swarms, such as schools of fish or flocks of birds. Windisch et al. proposed a method for performing structural testing using PSO and revealed that compared to genetic algorithm, this technique is much simpler, easier to implement, and possesses fewer parameters that the user needs to adjust [19].

Furthermore, a nature-inspired approach is the Ant Colony Optimization (ACO) approach, which we employed in our proposal. The use of ACO in test case generation has recently been proposed in several studies. To model the SUT, Srivastava et al. employed CFG. Ants traverse the SUT from the start node to the end node, leading to a combination of pheromone disposal and heuristic value at the edges, prioritizing those not yet visited. This leads to the completion of all paths coverage [20].

To effectively generate test sequences, considering the importance of states and the need to cover the most critical states, Srivastava et al. leveraged the statistical MBT based on the Markov chain, in addition to the ACO algorithm. In this method, the SUT model contains the probabilities of transitions between individual application states based on which test cases are effectively generated that match the desired importance. The proposed method yields good coverage of critical nodes using a small number of test sequences [21]. Sayyari and Emadi proposed a similar approach [22].

Let us describe this situation using our SUT model elements as explained in more detail in Section 4. 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 LCZ border nodes), they do not ensure the sequence of nodes required in the test cases. In contrast, our concept ensures this aspect. Notably, uncertainty exists concerning the LCZ IN node $n_{in} \in in(G, threshold)$ preceding the reachable LCZ OUT node $n_{out} \in out(G, threshold)$ in zone $L \in \mathcal{L}(G, threshold)$. Therefore, we cannot guarantee the fulfillment of the selected coverage criterion using such a technique. Our proposed approach primarily focuses on fulfilling these criteria, which renders it a novel contribution among the established path-based testing techniques.

2.4. *Alternative techniques for testing IoT system functionality with a limited network connection*

Muthiah and Venkatasubramanian focused on network connectivity testing and introduced the term connectivity testing [23]. Murad et al. mentioned the need to perform these connectivity tests in their study focusing on the healthcare industry [24]. Sirshar et al. used the same term in their preprint [25]. Furthermore, Esquiagola et al. performed connectivity tests on IoT platforms [26].

Alternatives to our approach, which focus on system functionality, primarily focus on the lower levels of an SUT, typically at the network level [27–29]. As a process-based viewpoint is insufficiently covered in the current literature, this makes the approach presented in this study an original contribution to the current research field.

The topic has been explored primarily by existing studies on quality of service (QoS) testing [3,28,30]. These studies tended to assess the reliability of a network, and higher levels of SUT functionality were not tested. However, testing IoT functionalities remains underexplored from a behavioral perspective.

In 2017, White et al. analyzed 162 research articles for a systematic mapping of state-of-the-art techniques employed in QoS approaches for IoT [28]. They discovered that the physical, link, and network layers of the OSI model were the most researched layers of IoT infrastructure as yet. However, the fields of deployment, middleware, and cloud layers required further research - the current research mainly concentrates on the individual layers of the IoT system infrastructure, among which the lower levels are better covered. In the current research, the high-level process viewpoint of the system, including integration and end-to-end testing, is rather neglected.

Moreover, testing system functionality under limited network connectivity is influenced by the volume and frequency of data exchange, typically for sensors and actuators in an IoT system. This is affected by the selected data-compression technique [31,32] and the use of methods for aggregating the data measured via sensors, thereby reducing the amount of communication required [33–35].

These mechanisms improve the overall energy efficiency of the sensor network, which allows for frequent repeated communication in case of a network failure and restoration [36,37].

Rudeš et al. conducted a study in this direction to present a concrete example of QoS assurance for IoT systems [27]. In their study, they tested a prototype of a small sensor network that shares its data with a server in the lab over the Internet. However, the tests were aimed only at the quality of network communication and not its influence on the overall process. Moreover, Matz et al. reported an analysis of quality assurance for network communication between IoT systems in the physical and application layers [30]. The authors measured the quality of a narrowband-IoT technology that enables energy-efficient and long-range network access to IoT devices on a cellular network (e.g., LTE or 5G in the future). Furthermore, Kim et al. proposed a service-based automatic IoT testing framework to resolve constraints on the coordination, costs, and scalability issues of traditional software testing [38]. In particular, this framework performs remote-distributed interoperability, scalable, automated conformance, and semantic testing. The set of test cases on the SUT is predefined; thus, exhaustive process testing cannot be achieved under a limited network connection.

The analyzed studies in this area assessed network reliability and related topics. Higher levels of SUT functionality (functional correctness from the system user's viewpoint or flawless integration) are not tested; therefore, techniques for testing IoT functionality are underexplored from the perspective of system behavior. No specific path-based or data flow techniques directly address the goals of this study. This conclusion was also confirmed by our recent independent systematic mapping study on the aspects of quality assurance in IoT systems [3]. The development of specific test-design techniques to test IoT systems operating with limited network connection [3] is a subject area that has not been explored sufficiently. Thus, considering the gap in literature, this subject area was aimed at in the present study. Hence, as process-viewpoint testing of the functionality of an IoT system under limited network connectivity remains unexplored and, as explained in this section, established path-based techniques are suboptimal for this purpose. Considering these aspects, we propose a new alternative technique in our study.

2.5. Summary and motivation

In IoT systems, especially mission-critical systems, reliable and safe system functionality must be ensured. This reliable and safe operation must also be guaranteed during network outages and restoration situations. As analyzed in Section 2, this area has not been adequately explored.

Although no unified definition exists for the system types belonging to the IoT family, all of these systems certainly employ the Internet as the connecting element, which is essential for operating the individual components of the system. Considering the use cases of individual IoT systems, wireless networks are primarily used to connect these components [39,40]. A wireless network can experience a connection outage owing to the users' mobile nature. Such situations involve mobile devices in locations with inadequate network coverage (e.g., uninhabited areas, tunnels, and subways), or when an energy shortage occurs in the communication infrastructure, or occasional hardware defects.

Tests for limited network connectivity should be approached from a process perspective, as performed in this study, owing to a number of reasons. First, a potential combinatorial explosion occurs in the number of use cases owing to the number of possible models and versions of the individual parts of the IoT system created in the general manner [41]. Second, smart device testers may need to test their devices in the real world, outside the lab, where Internet connectivity may be intermittent. Any test performed in this manner requires considerable time and resources. This application scenario favors the MBT approach because this algorithm can automatically generate a precisely optimized set of test cases using only the most important cases. Third, the demand for automating the IoT system-testing process is rising [8,42], and the process perspective yields an excellent basis for the approach presented herein.

Currently established path-based testing techniques and algorithms do not address the specificity of the problem model considered in this study (see Section 4).

This model contains several limited connectivity zones (LCZs) that represent a system undergoing network disruption. These LCZs are connected to the stable parts of the IoT system through the LCZ IN and LCZ OUT nodes, and they must be present in the test cases. Moreover, placing the LCZ IN node before the LCZ OUT node in t is essential for covering the pair of LCZ IN and OUT nodes on the LCZ border z in a test case t. The sequence of nodes between the LCZ IN and LCZ OUT nodes in t must not leave z. This rule is described in the coverage criteria introduced in Section 5.

This specific rule was not observed in the path-based testing techniques that were investigated. The concept of *test requirements* is the only exception in the field that can be utilized, and we have used this option in our proposal. As explained in Section 7.3 in detail, we transformed the coverage criteria *EachBorderOnce* and *AllBorderCombinations* (defined in Section 5) into the test requirements proposed by Li et al. by using their set cover algorithm to generate a set of test cases that satisfy these requirements [9]. However, apart from the algorithm by Li et al., we could not identify a more recent suitable algorithm that could be used in related work.

3. Principle of the Limited Network Connectivity Test (LNCT)

In the testing process, test designers construct a set of test scenarios for an IoT system to verify whether its functionality is affected by limited network connectivity or outage. The process focuses on testing the following two principal scenarios [43]:

(1) The network connectivity is interrupted in one part of the process handled by the SUT (or it is limited to an extent that affects proper functionality of the SUT). In such cases, testing the SUT functionalities should include scenarios such as[1]

[1] The given scenarios are only examples, and the list may not be complete; certain scenarios may not be relevant to all types of IoT systems.

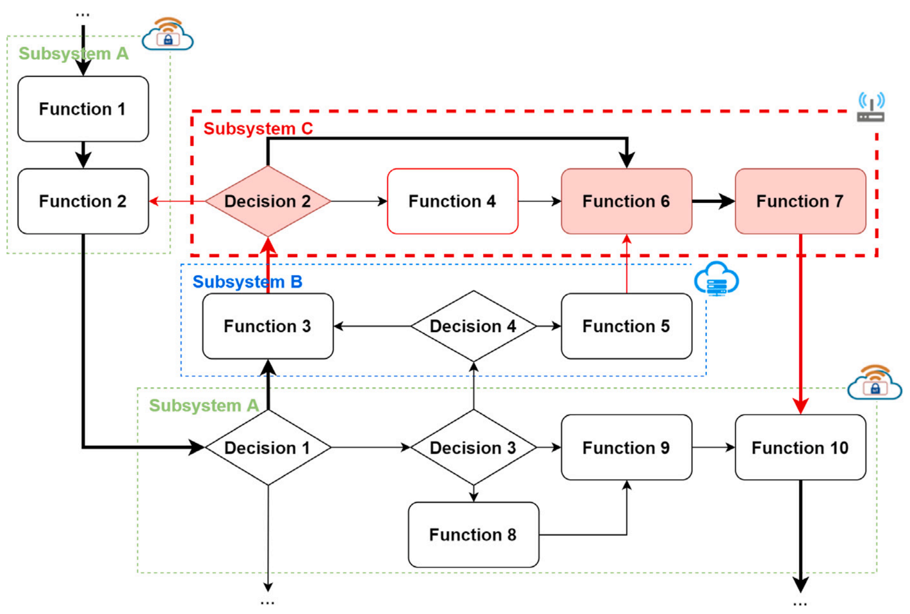


Fig. 1. Initial example of an SUT affected by network connectivity outage in its module and a path-based test case enabling the test of the SUT behavior in such a scenario.

(1) Assume that an SUT subsystem is isolated from network connectivity for a certain time when collecting data. Are these data correctly transmitted and stored offline when the network connectivity is available again, or are the collected data lost? (2) The SUT subsystem accepts signals (e.g., commands or API calls) from other devices or subsystems. This receiving subsystem is temporarily unconnected to the network. Are other devices sending signals to properly notify the missing (offline) subsystem that cannot react to these signals? (3) Is an SUT user notified properly such that the functionality may be limited for a certain time period because of a network connectivity outage?

(2) The network connectivity is restored after an outage. The following typical scenarios may be tested when network connectivity is available again:

(1) If SUT data must be processed transactionally, can this transactionality be maintained despite a network connectivity outage? Will the affected transactions be discarded correctly or completed via available caches when connectivity is restored? Are the cached transactions completed correctly, including the logical order of their steps? (2) Are these cached data transmitted to receive SUT modules correctly when network connectivity is restored when SUT devices, modules, or subsystems cache the data during a connectivity outage? Is such a temporary decrease acceptable to users and system safety, although this transmission may affect the responsiveness or performance of the SUT? (3) Are the data stored and processed by the SUT consistent when the network connectivity is restored, and are the locally stored data transmitted or transactions completed? (4) Is the user of the SUT adequately informed that the disabled functionality will be available again?

In this study, we approached the problem from the perspective of process testing (or path-based testing). The principle of the proposed technique involves executing a process in an SUT to test its functionality when affected by a network connection outage or limitation. Path-based test cases are constructed to test the aforementioned situations. We follow events when network connectivity is interrupted (or limited) by events wherein connectivity is restored [43]. An example of this situation is illustrated in Fig. 1.

Fig. 1 shows a sample of a fictional IoT system comprising three subsystems (devices and back-end systems). Subsystems A and C represent IoT devices: Subsystem A is connected to a stable network, whereas subsystem C is a mobile device operating in areas where network connectivity may be limited (e.g., rural, maritime, or subterranean areas). Subsystem B is the back-end part of the system connected to a stable network, and subsystem A handles two separate parts of the process.

In the test, we assume that Subsystem C operates without network connectivity. Furthermore, we exercise various process-flow variants to learn the system behavior under such a restriction. In our example, Functions 4, 6, and 7 and Decision 2 are influenced by network connectivity outages, and they are depicted with a red background. We need to exercise a transition from Function 3 to Decision 2, Decision 2 to Function 2, Function 5 to Function 6, and Function 7 to Function 10 to test the outlined scenarios (these transitions are indicated in red in Fig. 1). Fig. 1 illustrates (in bold arrows) an example of a test case that sequences the event when the network connectivity is interrupted or limited (arrow incoming to Decision 2) with an event in which the connectivity is restored (arrow outgoing from Function 7).

Various test paths that chain the events of network connectivity outages with connectivity restoration events can be composed using an SUT model. However, most test cases are not optimal considering the overall testing cost. Therefore, this study focuses on the generation of optimal test sequences that address the limited network connectivity problem.

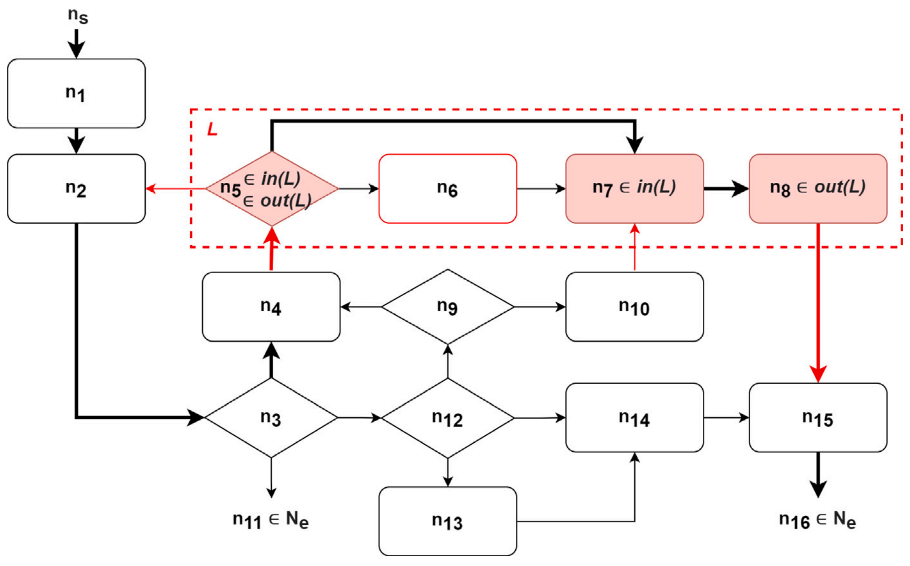


Fig. 2. Illustration of defined model elements in the introduced IoT system example.

4. Problem model

We generate test cases for the **SUT process** that can be impacted by a possible network connection outage (CO), which is abstracted as a directed graph $G = (N, E, n_s, N_e)$, where $N \neq \emptyset$ and $E \subseteq N \times N$ represent a finite set of nodes and a nonempty set of edges $e \in E$, respectively. Node $n_s \in N$ represents the initial/start node of graph G, $N_e = \{n_e \mid n_e \in N$ with no outgoing edge$\}$, defining a non-empty set of end nodes of graph G. The nodes serve as an abstraction of SUT actions, functions, or decision points, and the edges are the transitions between them in the process flow. In the proposed method, G does not allow parallel edges.

Test case t represents a sequence of nodes $n_1, n_2, \dots, n_n$ with a sequence of edges $e_1, e_2, \dots, e_{n-1}$, where $e_i = (n_{i,} n_{i+1})$, $e_i \in E$. The test case t starts with the start node n_s $(n_1 = n_s)$ and ends with the G end node $(n_n \in N_e)$. Test set T represents a set of test cases.

The **edge connection outage probability** (COP) denoted by $cop(e)$ is defined for $e \in E$, and represents the percentage number indicating the abstract probability of a connection outage at this edge. If $cop(e)$ is established, e is a transition in the IoT system process influenced by possible limited network connectivity.

The **threshold COP**, denoted by $threshold$ indicates the threshold connection outage probability for which the test set T is created. By setting $threshold$ to n, we assume that all edges with an LC probability greater than or equal to n are affected by a hypothetical tested connection outage.

We used the limited connectivity zone (LCZ) concept as follows: the **LCZ edge** is an edge $e \in E$ for which $cop(e) \geq threshold$, and the **non-LCZ edge** is an edge $e \in E$ for which $cop(e) < threshold$. The **LCZ** L is a coherent subgraph of G comprising only the LCZ edges, and G can contain more than one LCZs, denoted by $\mathcal{L}(G, threshold)$ for a specific $threshold$.

The **IN node** of LCZ L is node n that satisfies one of the following conditions:

1. $n = n_s$ and n possess an outgoing edge that is an LCZ edge of L.
2. n possess an outgoing edge that is an edge of L and 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 $\mathcal{L}(G, threshold)$.

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

1. $n \in N_e$ and n possess an incoming edge that is an edge of L.
2. n has an incoming edge that is an LCZ edge of L and 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 $\mathcal{L}(G, threshold)$.

The **border node** of the LCZ is either an IN or OUT node of the LCZ.

The concepts defined above are illustrated in Fig. 2 using the fictional IoT system described in Fig. 1 as a running example. In this example, $N_e = \{n_{11}, n_{16}\}$; L comprises 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 $t = n_s, n_1, n_2, n_3, n_4, n_5, n_7, n_8, n_{15}, and n_{16}$.

The LNCT test case-generation problem is summarized as follows: given an SUT model, G, $threshold$, and a test coverage criterion C (specified in Section 5), we determine a test set T that satisfies C.

Table 1
Test case evaluation criteria $\mathcal{E}$.

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

5. Test coverage criteria

Various test-coverage criteria C can be defined for the limited connectivity problem. In this study, we employed two test-coverage criteria: *EachBorderOnce* and *AllBorderCombinations*.

A test set T must contain each node of $in(G, threshold)$ and $out(G, threshold)$ to satisfy the ***EachBorderOnce*** criterion. 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 path but not necessarily immediately. The proposed technique allows n_{in} to be equal to n_{out}, considering that in certain scenarios, the LCZ may be entered and exited through the same node. In practice, this test coverage criterion requires that all LCZ border nodes be visited at least once in certain test cases, regardless of the manner in which the IN nodes are entered and the OUT nodes are exited.

The second coverage criterion, ***AllBorderCombinations***, requires that 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} within 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 a node $n_{out} \in out(L)$ later in the path but not necessarily immediately. Notably, n_{in} is equal to n_{out}. Thus, this test coverage criterion requires that all possible combinations of IN and OUT nodes of LCZ borders, for which a path exists inside the LCZ from an IN node to an OUT node, be visited by the test cases, regardless of which edges we enter the IN and leave the OUT nodes in the test case. sfy the ***ComprehensiveAllBorderCombinations***, 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 exists a path from n_{in} to n_{out} inside the L. Further, n_{in} must be entered by a non LCZ edge in some $t \in T$ and n_{out} must be exited by a non LCZ edge in this t.

6. Test set evaluation criteria

Several evaluation criteria for path-based testing have been discussed in literature [9,44,45]. We employed the options summarized in Table 1 for the problem discussed in this study as a test-set evaluation criterion $\mathcal{E}$.

Considering length dispersion ($s(T)$), this criterion aims to prevent excessively long and short test cases. Test analysts consider long test cases impractical because the probability that the test case is interrupted by a defect and the rest of the test case cannot be finished increases with the length.

In the experiments, we utilized $\mathcal{E}$ to evaluate the properties of T created by the algorithms discussed for the different SUT models.

7. Proposed algorithms

We compare three algorithms that generate T from G for the LNCT technique. The first two algorithms are our proposed approaches, and the third one is based on an established algorithm:

1. The **shortest paths composition algorithm (SPC)** is based on the principle of finding the shortest paths between identified points in G and chaining them in the test cases (description follows in Section 7.1),
2. The **ant colony optimization-based algorithm (ANT)** is based on the general ant colony optimization principle for determining the optimal test cases (description is given in Section 7.2),
3. The **enforced prime paths algorithm (EPP)** employs a previous path-based algorithm that supports the test requirements (description is provided in Section 7.3).

7.1. Shortest Path Composition algorithm (SPC)

The main routine of SPC is defined in Algorithm 1, which accepts G, $threshold$, and C as inputs and produces T as an output while maintaining a set of unvisited LCZ IN nodes U_{in} and a set of unvisited LCZ OUT nodes U_{out}. Algorithm 1 starts by determining the shortest paths between $in(L)$ and $out(L)$ within all LCZ $L \in \mathcal{L}(G, threshold)$ of the given G, which is achieved using the subroutine $FindPathsInsideLCZs$ described in Algorithm 2.

The subroutine $FindPathsInsideLCZs$ operates on the breadth-first search principle, starting in the $out(L)$ nodes of each LCZ $L \in \mathcal{L}(G, threshold)$. The nodes that must be traversed are stored in queue Q. The distance from a particular node $n_{out} \in out(L)$ is stored for all the explored nodes; this distance is denoted by $distance(n_1, n_2)$. For each L, the paths of each $in(L)$ with the shortest distance to $out(L)$ are selected as the output, which is denoted by P. In Algorithm 2, $parents(n)$ denote the set of parents of node n.

The main routine of Algorithm 1 is to continue by exploring G using the breadth-first search principle. The exploration history is expanded by p when this search reaches the starting node of any path $p \in P$. Then, the end node of p is added to the set of nodes from which further exploration of the graph is conducted. This behavior is performed using the $GetNextShortestPathInLCZ$ procedure described in Algorithm 3. For *AllBorderCombinations*, this procedure removes p from P, and for *EachBorderOnce*, all paths p' ending with the already visited LCZ OUT nodes (nodes that are not in U_{out}) are removed from P. We can remove p' from P if there exists any subpath p' of p connecting some unvisited LCZ IN nodes to the LCZ OUT node.

Further exploration of G continues to reach the start node of another $p \in P$, which then follows by repeating the behavior described above. A test case is composed based on the exploration history and added to a set of test cases T produced as the output of the algorithm if an end node of G is reached during exploration.

Algorithm 1: $SPC(G, threshold, C)$**:** Identify all relevant shortest paths inside the LCZs and between LCZs of G, start node and end nodes of G, then combine them in the test cases

```
   Input  : SUT model $G$, $threshold$, and coverage criterion $C$
   Output: set of test cases $T$
 1 $T \leftarrow \emptyset$, $U_{in} \leftarrow in(G, threshold)$, $U_{out} \leftarrow out(G, threshold)$
 2 $P \leftarrow FindPathsInsideLCZs(G, threshold)$
 3 while $P \neq \emptyset$ do
 4     PUT $n_s \in G$ to $Q$ ;                                   ▷ $Q$ is a queue of nodes to traverse
 5     set $path$ as empty ;                                       ▷ $path$ is a sequence of nodes
 6     while $Q$ is not empty do
 7         $n \leftarrow$ POP from $Q$ ;                           ▷ $n$ is a currently traversed node
 8         if $n \in in(G)$ and $P$ contains a path that starts in $n$ then
 9             $p \leftarrow GetNextShortestPathInLCZ(C, P, n, U_{in}, U_{out})$
10             $P \leftarrow P \setminus \{p\}$
11             $o \leftarrow$ the last node of $p$ ;               ▷ $o \in out(G, threshold)$
12             for each $n_p \in p$ do
13                 $potential\_previous\_TC\_step(n_p) \leftarrow parent(n_p)$
14             end
15             SET $Q$ as empty, PUT $o$ to $Q$
16         end
17         else if $n \in N_e$ then
18             $t$ is a path containing only $n$, $temp \leftarrow n$
19             while $potential\_previous\_TC\_step(temp)$ has been set do
20                 add $potential\_previous\_TC\_step(temp)$ at the beginning of $t$
21                 $temp \leftarrow potential\_previous\_TC\_step(temp)$
22             end
23             $T \leftarrow T \cup t$
24         end
25         else
26             for each $d \in descendants(n)$ do
27                 $potential\_previous\_TC\_step(d) \leftarrow n$
28             end
29         end
30     end
31 end
32 return $T$
```

Algorithm 2: $FindPathsInsideLCZs(G, threshold)$: 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 $P$
$P \leftarrow \emptyset$
for each LCZ $L \in \mathcal{L}(G, threshold)$ do
    for each $n_{out} \in out(L)$ do
        SET $Q$ as empty ;                                    ▷ $Q$ is a queue of nodes to traverse
        PUT $n_{out}$ to $Q$
        for each $x$ in $L$ except $n_{out}$ do
            $distance(n_{out}, x) \leftarrow \infty$ ;        ▷ distance equals to number of nodes of a path from $n_{out}$ to $x$
        end
        while $Q$ is not empty do
            $n \leftarrow$ POP from $Q$
            for each $p \in parents(n)$ do
                if $distance(n_{out}, p) > distance(n_{out}, n)$ then
                    $distance(n_{out}, p) \leftarrow distance(n_{out}, n) + 1$
                    PUT $p$ to $Q$
                end
            end
        end
        for each $n_{in} \in in(L)$ do
            SET $path$ as empty ;                             ▷ $path$ is a sequence of nodes
            $n \leftarrow n_{in}$ ;                           ▷ $n$ is a currently traversed node
            while $n \neq n_{out}$ do
                $n \leftarrow x \in L$ such that $distance(n, x)$ is minimal
                ADD $n$ at the end of $path$
            end
            ADD $n$ at the end of $path$
            if $|path| > 1$ then
                $P \leftarrow P \cup path$
            end
        end
    end
end
return $P$
```

7.2. Ant-colony-optimization-based algorithm (ANT)

The ANT algorithm uses the ACO principle introduced by Dorigo [46]. According to this principle, a specific algorithm was formulated to solve the discussed problem. The main routine of ANT is described in Algorithm 4, which accepts G, $threshold$, and C as inputs, and produces T as the output. The following section presents the algorithm variables, their initiation, algorithm steps, and details on how the desirability levels are obtained and the best ant is selected.

7.2.1. Algorithm variables

The following variables are used in the ANT algorithm.

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

During the experiments, the following values of the variables were found after extensively fine-tuning the algorithm to yield the best results for the ANT algorithm: $\alpha = 1$, $\beta = 3$, $NC = 10$, $\rho = 0.5$, $Q = 1.0$, $m = 50$, and $c = 1.0$.

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

```
Input  : coverage criterion $C$, 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 $p$
1  $n_{out} \leftarrow$ a node to which there exists a path from $n_{in}$ present in $P$
2  if $C = EachBorderOnce$ then
3      if $n_{out} \notin U_{out}$ and $P$ contains a path that ends in an $n'_{out} \in U_{out}$ then
4          $n_{out} \leftarrow n'_{out}$
5      end
6  end
7  $p \leftarrow$ a path from $n_{in}$ to $n_{out}$ that is present in $P$
8  $P \leftarrow P \setminus \{p\}$
9  for each $n \in p$ do
10     for each path $p'$ from $n$ that is present in $P$ do
11         $n'_{out} \leftarrow$ the end node of path $p'$
12         if $n'_{out}$ follows $n$ in the path $p$ then
13             $P \leftarrow P \setminus \{p'\}$
14         end
15     end
16 end
17 if $C = EachBorderOnce$ then
18     $P \leftarrow P \setminus \{ p \mid p \in P,\ p$ starts in $n_{in}$, $p$ leads to an $n_x \notin U_{out}$ $\}$
19 end
20 return $p$
```

7.2.2. Algorithm initiation

The main ANT routine is specified in Algorithm 4. The algorithm accepts G, $threshold$, and C as inputs, and produces T as an output. In the initial step, we create a map $\mathcal{U}$ that contains a set of reachable nodes $U \subset out(L)$ for every node $n \subset in(L)$. This map is constructed for all LCZs L.

In Algorithm 4, the out node $r \subset U$ that is reachable from node n indicates that a directed path exists from n to r. The method for generating $\mathcal{U}$ traverses L. This traversal begins at the $n_{in} \in in(L)$ nodes and ends at the $n_{out} \in out(L)$ nodes using the BFS algorithm.

The ANT algorithm continues by repeatedly calling the *ANTCore* procedure (see Algorithm 5), which produces a path that contains the LCZ border-node pairs that are then removed from the map $\mathcal{U}$. Algorithm 4 continues until the map $\mathcal{U}$ is not empty.

7.2.3. The ANTCore algorithm

The *ANTCore* procedure described in Algorithm 5 manages the traversal of ants through G. The ants were led by a combination of desirability and pheromone disposal at the edges of G. In this algorithm, we created a list of m ants. The map of desirabilities H is initialized by calling the $InitDesirabilities$ procedure specified in Algorithm 6, which is further described in Section 7.2.5. Algorithm 6 initializes the mapping $\mathcal{D}$, which for each node $x \in N \in G$ returns the LCZ border nodes that are reachable (mapping $\mathcal{D}$ returns a set (R_{in}, R_{out}), where the reachable LCZ IN nodes are stored in the set R_{in} and the reachable LCZ OUT nodes in the set R_{out}).

7.2.4. Beginning of the ant's traversal

We start by traversing G at node n_s for each ant $t \in A$. We initialize temporary variables, such as the t path, stored in P, or sets of the LCZ IN and LCZ OUT nodes B_{in} and B_{out} that t covers. The traversal of G by t is driven by a combination of pheromone elimination and the level of desirability at the edges.

7.2.5. Obtaining the desirability levels

The process of obtaining the desirability levels stored on a map H begins with the execution of Algorithm 6, which, for each $n_e \in N_e$, uses the $GenerateReachableBNs$ procedure specified in Algorithm 7 to create the mapping $\mathcal{D}$, introduced in the previous section. Algorithm 7 traverses G from the end nodes to the start node using the DFS algorithm, which stores the LCZ IN and LCZ OUT nodes that are reachable from all nodes of $N \in G$.

7.2.6. Traversing G and covering LCZ border nodes

After calculating the desirabilities of the edges, we calculated the probabilities of the moving ant from its current location $i \in N$ to a neighboring node $j \in N_i$ using edge (i, j), which was inspired by Dorigo's formula (1) in [47]. Using this probability, the ant selects the node $j \in N_i$ to which it moves. We use the procedure $ManageCoveredBorderNodes$ described in Algorithm 9 to update

Algorithm 4: $ANT(G, threshold, C)$: Main routine of the ANT Algorithm which explores G via ants and maintains a map of covered LCZ border nodes $\mathcal{U}$

```
Input  : SUT model $G$, $threshold$, coverage criterion $C$,
Output: Set of test cases $T$
1  $\mathcal{U}$ is a map 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 can be reached 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      $(a, \mathcal{P}, \mathcal{B}_{in}, \mathcal{B}_{out}) \leftarrow ANTCore(G, threshold, C, \mathcal{U})$
6      $T \leftarrow T \cup \{\mathcal{P}[a]\}$
7      for each $n_{in} \in \mathcal{B}_{in}[a]$ do
8          for $n_{out} \in \mathcal{B}_{out}[a]$ do
9              REMOVE $n_{out}$ from $\mathcal{U}[n_{in}]$
10         end
11     end
12     for each $n_{in} \in \mathcal{B}_{in}[a]$ do
13         if $\mathcal{U}[n_{in}] = \emptyset$ then
14             REMOVE a key $n_{in}$ from $\mathcal{U}$ ;          ▷ Remove in $\mathcal{U}$ LCZ IN node covered by $a$
15         end
16     end
17 end
18 return $T$
```

Algorithm 5: $ANTCore(G, threshold, C, \mathcal{U})$ *(1st part)*:
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 $C$, a map of uncovered nodes $\mathcal{U}$
Output: Ant's champion $a$, who found the best path among the others, $\mathcal{P}$, which is a map of ant paths, where key is an ant
         and value is its path, $\mathcal{B}_{in}$ and $\mathcal{B}_{out}$, which are maps of LCZ IN and LCZ OUT, nodes covered by ants, where key is
         an ant and the values are the nodes covered by this ant.
1  Set the constants for the ANT algorithm defined in 7.2.1 to: $\alpha = 1$, $\beta = 3$, $NC = 10$, $\rho = 0.5$, $Q = 1.0$, $m = 50$, and $c = 1.0$.
2  $A$ is a set $m$ ants (see Section 7.2.1)
3  $\mathcal{D}$ 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 $x$.
4  $\mathcal{D} \leftarrow InitDesirabilities(G,\ threshold,\ \mathcal{U})$ ;          ▷ Find LCZ border nodes reachable from each $n \in N \in G$
5  $count \leftarrow 0$
6  ... continues on the next page
```

the sets B_{in} and B_{out} of the covered LCZ IN and LCZ OUT nodes, respectively, based on the current path of the ant and the nature of node j (e.g., the LCZ IN node, uncovered LCZ OUT node, or other types).

7.2.7. Choosing the best ant

The algorithm selects the best ant $a \in A$ (champion) when all ants finish their paths and find an end node $n_n \in N_e$. The process of finding the champion is specified in Algorithm 10, which iterates A and selects the best ant based on the found path. The best path contains the highest number of LCZ border nodes that are not yet covered and has the shortest length.

7.2.8. ANTCore repetitions

The steps described in Sections 7.2.4 to 7.2.7 are repeated NC several times so that the pheromone levels have a more significant effect. After completing the repetition, we select the champion $a \in A$ again using the $FindChampion$ procedure (Algorithm 10) and return it to the main ANT routine (Algorithm 4). The map of the uncovered LCZ border nodes $\mathcal{U}$ is updated based on a's path and the LCZ border nodes that it contains.

7.3. Enforced Prime Path algorithm (EPP)

The EPP algorithm first creates a set of test requirements to satisfy the test coverage criteria C. Then, it uses an existing set-covering algorithm for prime path search to find T.

Algorithm 5: *$ANTCore(G, threshold, \mathcal{C}, \mathcal{U})$ (2nd part)*

```
6  while $count \leq NC$ do
7      $count \leftarrow count + 1$
8      for each $t \in A$ do
9          Place ant $t$ to position $n_s$
10         $P \leftarrow$ empty path ;                    ▷ $P$ is a path of ant $t$
11         $P_L \leftarrow$ empty path ;                  ▷ $P_L$ is a path of ant $t$ through an LCZ $L \in \mathcal{L}(G, threshold)$
12         $B_{in} \leftarrow \emptyset$ ;                ▷ A set of LCZ IN nodes covered by ant $t$
13         $B_{out} \leftarrow \emptyset$ ;               ▷ A set of LCZ OUT nodes covered by ant $t$
14         $n_{in} \leftarrow nil$ ;                      ▷ The last LCZ IN node reached by ant $t$ during its path
15         while $t$ has not reached $n_e$ do
16             $i \leftarrow$ current position of $t$ ; $N_i \leftarrow$ set of $i$ descendants ;    ▷ $i$ is a currently iterated node
17             $H \leftarrow CalculateDesirabilities(G, \mathcal{U}, \mathcal{C}, i, n_{in}, P, \mathcal{D}, B_{in}, B_{out})$
18             $E_{N_i}$ is a map where a value is a number $< 0, 1 >$ of the probability that the ant $t$ moves to $j \in N_i$ through the
                 edge $x$ incoming to $j$ and the key is $x$
19             for all nodes $j \in N_i$ do
20                 $E_{N_i}[(i,j)] \leftarrow \frac{(\tau_{ij})^\alpha \cdot (H[(i,j)])^\beta}{\sum_{l \in N_i} \tau_{il} \cdot H[(i,l)]}$ ;  ▷ Store the probability of ant $t$ moving to node $j$ by edge $(i, j)$
21             end
22             Randomly select the next node $j \in N_i$ to move the ant $t$ in, using edge $(i, j)$. In this selection, the probability
                 $E_{N_i}[(i, j)]$ is used.
23             if $cop((i, j)) \geq threshold$ then
24                 Add $(i, j)$ at the end of $P_L$
25             end
26             Add $(i, j)$ at the end of $P$ ; Move $t$ from node $i$ to the neighboring node $j$ using edge $(i, j)$
27             if $j$ is a key in $\mathcal{U}$ then
28                 $n_{in} \leftarrow j$ ;                ▷ node $j$ is uncovered LCZ IN node
29             end
30             else if $j$ is anywhere in values of $\mathcal{U}$ then
31                 $B_{in} = B_{in} \cup \{n_{in}\}$, $B_{out} = B_{out} \cup \{j\}$
32                 for each node $x$ in $P_L$ from the beginning of $P_L$ do
33                     $P'_L \leftarrow$ part of $P_L$ starting with $x$
34                     for each node $p'$ in $P'_L$ from the beginning of $P'_L$ do
35                         $(B_{in}, B_{out}) \leftarrow ManageCoveredBorderNodes(x, p', B_{in}, B_{out}, \mathcal{U}, \mathcal{C}, P_L)$
36                     end
37                 end
38             end
39         $\mathcal{P}[t] \leftarrow P$, $\mathcal{B}_{in}[t] \leftarrow B_{in}$, $\mathcal{B}_{out}[t] \leftarrow B_{out}$
40     $a \leftarrow FindChampion(A, \mathcal{B}_{in}, \mathcal{B}_{out}, \mathcal{P})$ ;    ▷ Select the champion $a$ from the set of ants $A$
41     for each edge $(i, j)$ in $\mathcal{P}[a]$ do ;    ▷ $|\mathcal{P}[a]|$ denotes the number of nodes in path
42         $\tau_{ij} \leftarrow \tau_{ij} + \frac{1}{|\mathcal{P}[a]|}$ ;    ▷ Deposit pheromone on edges that were used, for $\tau_{ij}$ refer to Section 7.2.1
43     end
44     for each $(i, j) \in E \in G$ do
45         $\tau_{ij} \leftarrow (1 - \rho) * \tau_{ij}$ ;    ▷ Pheromone decay of all edges, for $\rho$ refer to Section 7.2.1
46     end
47 $a \leftarrow FindChampion(A, \mathcal{B}_{in}, \mathcal{B}_{out}, \mathcal{P})$ ;    ▷ The best in the last iteration
48 return $(a, \mathcal{P}, \mathcal{B}_{in}, \mathcal{B}_{out})$
```

The core of the EPP is based on a greedy set-covering algorithm, and it aims to solve the minimum-cost test-path problem by adopting approximation algorithms for the shortest superstring problem. This algorithm was introduced by Li et al. [9]. We have not yet identified a more recent algorithm that can solve the problem suggested in this study. The inputs to the greedy set-covering algorithm are a set of test requirements R and a small set of test paths TP, where a *test path* is a path in G. A set of *test requirements* is a set of elements of a system that must be covered by tests. In our case, it is a set of specific paths through G. The test paths are *prime paths*, which implies that they are simple and do not appear as sub-paths of other simple paths. Furthermore, *simple paths* have no internal loops, and only the first and last nodes of the path can repeat.

Algorithm 6: $InitDesirabilities(G, threshold, \mathcal{U})$: Traverse G and find which LCZ border nodes are reachable from each $n \in N \in G$

```
Input  : SUT model $G$, $threshold$ and a map of uncovered nodes $\mathcal{U}$
Output: Mapping $\mathcal{D}$ that for each node $x \in N \in G$ returns which LCZ border nodes can be reached from $x$
1 $\mathcal{D} \leftarrow \emptyset$
2 for each $n_e \in N_e$ do
3     $V \leftarrow \emptyset$ ;                                          ▷ A set to store visited nodes
4     $GenerateReachableBNs(G, threshold, \mathcal{U}, \mathcal{D}, V, n_e)$ ;      ▷ Fill the map $\mathcal{D}$
5 end
6 return $\mathcal{D}$
```

Algorithm 7: $GenerateReachableBNs(G, threshold, \mathcal{U}, \mathcal{D}, V, n)$: Traverse the SUT model G to find from which nodes we can reach which LCZ border nodes

```
Input  : SUT model $G$, $threshold$, a map 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 $p \in parents(n)$ do
2      if $p \notin V$ then
3          $V \leftarrow V \cup p$
4          $(R^p_{in}, R^p_{out}) \leftarrow \mathcal{D}(p)$
5          if $n \in in(G, threshold)$ and $n$ is a key in $\mathcal{U}$ then
6              $R^p_{in} \leftarrow R^p_{in} \cup n$
7          end
8          if $n \in out(G, threshold)$ and $\mathcal{U}$ contains a value $n$ for any key then
9              $e$ is an edge from $p$ to $n$
10             if $cop(e) \geq threshold$ then
11                 $R^p_{out} \leftarrow R^p_{out} \cup n$
12             end
13         end
14         $\mathcal{D}(p) \leftarrow (R^p_{in}, R^p_{out})$
15         $GenerateReachableBNs(G, threshold, \mathcal{U}, \mathcal{D}, V, p)$
16     end
17 end
```

7.3.1. The main algorithm

The main procedure of the EPP is described in Algorithm 11, it begins with a sub-routine for constructing a set of test requirements R defined in Algorithm 12, which iterates $\mathcal{L}$ and determines the shortest paths from all $x \in in(L)$ to all $y \in out(L)$. This step is only necessary if the *AllBorderCombinations* coverage criterion is selected. We added all of the shortest paths that we found to R. Otherwise, the algorithm reduces the set of paths found when *EachBorderOnce* coverage is selected. We sorted the paths by their lengths and stored them in a new list $\mathcal{X}$. Subsequently, we traverse $\mathcal{X}$ and add to R only those paths that contain LCZ IN or LCZ OUT nodes that have not yet been added to R.

7.3.2. Greedy set-covering algorithms

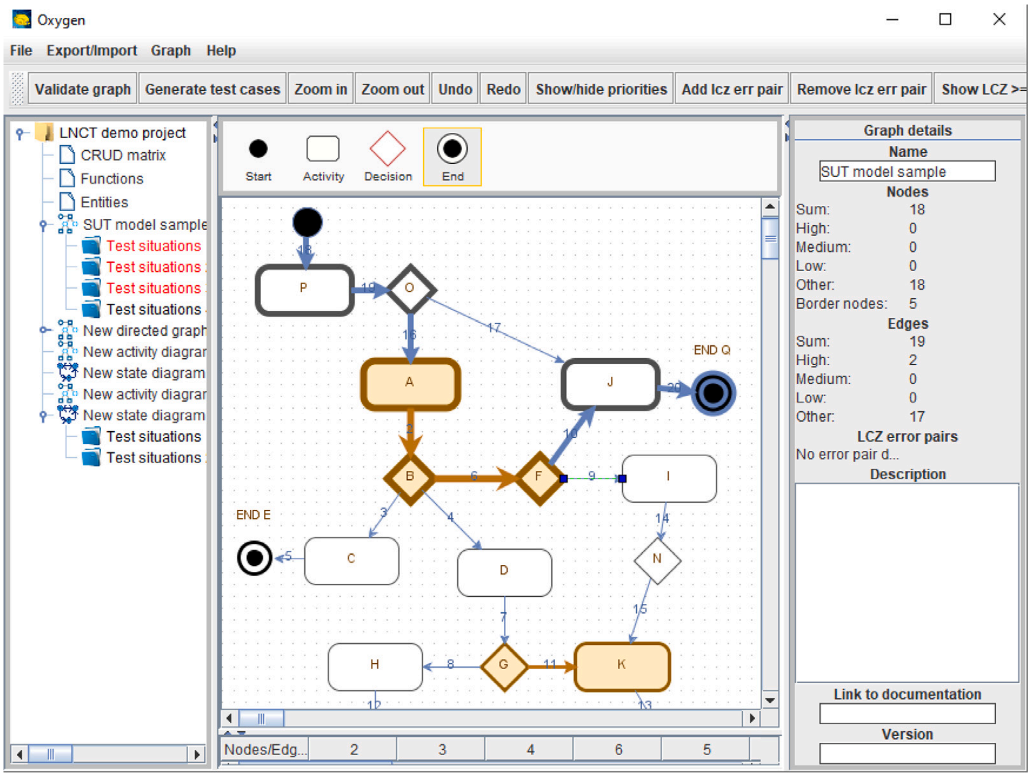
The set of test requirements R is an input to Li's $setCoveringAlgorithm$ procedure [9], chaining them together into one long path, called the super-test requirement $\prod$. The $getSmallSetOfTestPaths$ procedure traverses G and generates a set of all possible test paths, TP. Finally, TP, $\prod$, and G are inputs to Li's $splitSuperTestRequirement$ procedure [9] based on TP, which splits a super-test requirement $\prod$ into a set of final test paths T that starts in n_s and ends in one of $n_e \in N_e$. The selected test-coverage criterion is satisfied by T because the selected test-coverage criterion is reflected by the set of test requirements R.

8. Experimental evaluation

In the experiments, we compared the test cases created by the proposed SPC, ANT, and EEP algorithms for a set of 150 SUT models and test coverage criteria introduced in Section 5. The evaluation criteria defined in Section 6 were employed to compare the test cases. In this section, we illustrate the experimental method and its setup, and present the results of the experiments.

Algorithm 8: $CalculateDesirabilities(G, \mathcal{U}, C, i, n_{in}, P, \mathcal{D}, B_{in}, B_{out})$ *(1st part)*: Calculates desirabilities of the edges according to the number of not yet covered border nodes reachable from surrounding nodes.

```
Input : SUT model $G$, map of uncovered LCZ border nodes $\mathcal{U}$, coverage criterion $C$, 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 $B_{out}$,
Output: Map $H$ for ant-routing to neighbors of $i$ where, key is an edge and value is a desirability.
1  Set $H$ as empty
2  For each edge $t$ outgoing from node $i$ do
3      $h_t \leftarrow 0$ ;                 ▷ Number of reachable uncovered LCZ border nodes when using edge $t$
4      $j \leftarrow$ target node of edge $t$
5      $(R^j_{in}, R^j_{out}) \leftarrow \mathcal{D}(j)$ ;      ▷ Symbols defined in Algorithm 5
6      $R^j_{in}{}' \leftarrow \emptyset$ ;      ▷ Uncovered LCZ IN nodes reachable from $j$
7      $R^j_{out}{}' \leftarrow \emptyset$ ;      ▷ Uncovered LCZ OUT nodes reachable from $j$
8      if $j \in R^j_{in}$ then
9          $R^j_{in}{}' \leftarrow R^j_{in} \setminus B_{in}$
10     end
11     if $j \in R^j_{out}$ then
12         for each LCZ OUT node $r \in R^j_{out}$ do
13             if $r \notin B_{out}$ then
14                 $R^j_{out}{}' \leftarrow R^j_{out}{}' \cup \{r\}$
15             end
16         end
17     end
18 ... continues on the next page
```



Graph details

Name: SUT model sample

Nodes
Sum: 18
High: 0
Medium: 0
Low: 0
Other: 18
Border nodes: 5

Edges
Sum: 19
High: 2
Medium: 0
Low: 0
Other: 17

LCZ error pairs
No error pair d...

Description

Fig. 3. Oxygen application with an SUT model, highlighted LCZs, and a test case.

Algorithm 8: *CalculateDesirabilities*(G, $\mathcal{U}$, C, i, n_{in}, P, $\mathcal{D}$, B_{in}, B_{out}) *(2nd part)*

```
    if $t$ is a LCZ edge then
        $L$ is a LCZ which contains $t$
        if $n_{in}$ is not nil then
            $R^j_{out}{}' \leftarrow R^j_{out}{}' \setminus \{i\}$ ;          ▷ Because $i$ is uncovered LCZ OUT node
            $R^j_{in}{}' \leftarrow R^j_{in}{}' \setminus in(L)$
            $h_t \leftarrow 1 + |R^j_{in}{}'| + |R^j_{out}{}'|$
        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
            $h_t \leftarrow 1 + |R^j_{in}{}'| + |R^j_{out}{}'|$
        end
        else
            $R^j_{in}{}' \leftarrow R^j_{in}{}' \setminus out(L)$
        end
    end
    else if $n_{in}$ is not nil and ($i \in \mathcal{U}$ or $C = EachBorderOnce$) then
    end
    $h_t \leftarrow 1 + |R^j_{in}{}'| + |R^j_{out}{}'|$
    else if $n_{in}$ is nil and $i \in \mathcal{U}$ and $i \notin B_{in}$ then
        $h_t \leftarrow 0$ ;          ▷ To avoid reaching the border of LCZ zone
    end
    else
        $h_t \leftarrow |R^j_{in}{}'|$ ;          ▷ $t$ is leading to a LCZ OUT node
    end
    if $h_t > 0$ then
        $h_t \leftarrow \frac{1}{-2h_t} + 1$
    end
    else
        $h_t \leftarrow 0$
    end
    PUT $h_t$ to $H$ with key $t$
end
return $H$
```

8.1. Implementation of the algorithms

SPC, ANT, and EEP algorithms were implemented in Oxygen.[2] Oxygen is an MBT platform developed by our research group. It is an open-source freeware platform created in Java 1.8 [48]. We extended the graphical editor of the SUT model for the creation of G.

An example of an SUT model constructed with oxygen is depicted in Fig. 3. The figure presents the UML Activity Diagram of a Smart Home inspired by the system proposed by Aravindan et al. [49]. The central server of the Smart Home communicates over a network with three subsystems. The first is a database server, and nodes B - C - D - E - F - Q model a subprocess handled by this subsystem. The second subsystem constitutes a central IoT server (the sub-process handled by this subsystem is modeled by nodes H - I - J - K). The last subsystem is a Raspberry Pi with connected sensors and actuators (the subprocess provided by this subsystem is modeled by nodes N - O - P - $END\ T$). The probability of a network outage is higher than the *threshold* level when the central

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

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

```
Input  : LCZ IN node $n_{in}$ and LCZ OUT node $n_{out}$ to be covered, a set of covered LCZ IN nodes $B_{in}$ and LCZ OUT nodes $B_{out}$,
         a map $\mathcal{U}$ of uncovered LCZ border nodes, coverage criterion $C$, a set $P_L$ where a path through LCZ $L$ is stored
Output: Updated sets $B_{in}$ and $B_{out}$
if $n_{in} \in B_{in}$ then
    if $n_{out} \in B_{out}$ then
        $B_{out} = B_{out} \cup \{n_{out}\}$
        if $n_{in} = n_{out}$ and $n_{in} \in B_{out}$ and $|P_L| > 0$ then
            $B_{out} = B_{out} \cup \{n_{in}\}$
        end
    end
end
if $C = EachBorderOnce$ then
    for each $u_{out}$ anywhere in values of $\mathcal{U}$ do
        if $u_{out} = n_{out}$ then
            $B_{out} = B_{out} \cup \{n_{out}\}$
        end
    end
    $B_{in} = B_{in} \cup \{n_{in}\}$
end
else
    if $\mathcal{U}[n_{in}] \subset B_{out}$ then
        $B_{in} = B_{in} \cup \{n_{in}\}$
    end
end
return $(B_{in}, B_{out})$
```

Algorithm 10: $FindChampion(A, \mathcal{B}_{in}, \mathcal{B}_{out}, \mathcal{P})$: 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 $A$, map of LCZ IN nodes $\mathcal{B}_{in}$ visited by each ant, map of LCZ OUT nodes $\mathcal{B}_{out}$ visited by each ant, map
         of each ant paths $\mathcal{P}$
Output: Ant $a$ that found the most efficient path
$a \leftarrow$ any ant from $A$
for each $t \in A$ do
    if $(\mathcal{B}_{in}[t] + \mathcal{B}_{out}[t]) > (\mathcal{B}_{in}[a] + \mathcal{B}_{out}[a])$ then
        $a \leftarrow t$
    end
    else if $(\mathcal{B}_{in}[t] + \mathcal{B}_{out}[t]) = (\mathcal{B}_{in}[a] + \mathcal{B}_{out}[a])$ then
        if $|\mathcal{P}[t]| < |\mathcal{P}[a]|$ then
            $a \leftarrow t$
        end
    end
end
return $a$
```

server communicates with external subsystems. Therefore, LCZ zones were formed and visually separated from the rest of the graph by a light brown color. Additionally, the symbols of the LCZ IN and LCZ OUT nodes are light brown. In the left application panel, the P generated by the SPC algorithm is visible, and these are denoted as test situations.

A pop-up window opens with individual test cases when the user clicks on the item. This test case is visually highlighted in the model when the user selects test cases from the list. In the sample presented in Fig. 3, we can observe the highlighted test case (composed of nodes *START - A - B - F - C - G - H - J - R - P - O - END T*).

Part of the Oxygen platform (development version) includes a module for comparing algorithms, which we configured for this study. This comparison module allows multiple algorithms to be executed on a given set of SUT models (saved in oxygen format). When test cases are generated for an SUT model, the module computes the defined properties (herein, a set of evaluation criteria;

Algorithm 11: EPP(G, $threshold$, C): The main routine of the EPP algorithm which creates a set of test requirements R to tour through LCZs and using this set, it constructs T as a set of prime paths containing these test requirements

```
Input  : SUT model $G$, coverage criterion $C$, $threshold$
Output: set of test cases $T$
$R \leftarrow$ getTestRequirements($G$, $C$, $threshold$)
$\prod \leftarrow$ setCoveringAlgorithm($G$, $R$) ▷ Defined in [9]
$TP \leftarrow$ getSmallSetOfTestPaths($G$) ▷ Defined in [9]
$T \leftarrow$ splitSuperTestRequirement($G$, $\prod$, $TP$) ▷ Defined in [9]
return $T$
```

Algorithm 12: $GetTestRequirements(G, C, threshold)$: Construct a set of test requirements R that would be used in generation of T to tour all $L \in \mathcal{L}(G, threshold)$ so that C would be satisfied.

```
Input  : SUT model $G$, coverage criterion $C$, $threshold$
Output: set of test requirements $R$
$R \leftarrow \emptyset$
if $C = AllBorderCombinations$ 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 $C = EachBorderOnce$ 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 in ascending order of lengths
        $L_{in} \leftarrow in(L)$, $L_{out} \leftarrow out(L)$
        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
```

refer to Section 6). Subsequently, the module exports the results in a consolidated summary report in the CSV file, which enables further analysis and processing of the data.

8.2. Experiment method and set-up

To compare the SPC, ANT, and EEP algorithms, we prepared 150 models for different SUTs that varied by $|N|$, $|E|$, number of LCZs (denoted as $|\mathcal{L}|$), number of potential LCZ IN and OUT nodes (denoted as $|in(G)|$ and $|out(G)|$), number of cycles ($cycles$),

Table 2
Overall properties of SUT models used in the experiments.

	$\|N\|$	$\|E\|$	$\overline{deg(n)}$	$\|N_e\|$	$cycles$	$\|\mathcal{L}\|$	$\|in(G)\|$	$\|out(G)\|$	$\|\chi(G)\|$
MIN	19	26	2.39	1	0	1	1	1	1
MAX	442	606	4.26	21	54	4	18	18	32
$\overline{x}$	66.95	101.73	3.15	3.24	6.85	2.23	4.79	5.15	5.53
$\tilde{x}$	40	60.5	3.15	3	3	2	4	5	4

average node degree ($\overline{deg(n)}$), and $|N_e|$. Various methods were generated to create the SUT model. We considered 11 real projects with documented process models,[3] which we transformed into directed graphs with LCZs. We created 11 other models by relocating the LCZs in these models. We constructed another 98 models that varied according to the number of nodes, edges, and LCZs. These models do not represent certain existing systems. However, they are based on the topology of existing system models created to closely resemble their topology. In particular, they possess sufficient flexibility for creating sufficient model variants for experiments. We generated another 30 models using a specialized generator that we implemented to further extend the variability of the input set. The inputs to this generator are $|N|$, $|E|$, $|N_e|$, $cycles$, the number of LCZs $|\mathcal{L}|$, and for each LCZ L, the number of nodes, edges, cycles, $in(L)$, and $out(L)$. G is generated as the output.

We extend the SUT models by simulating the defects caused by limited network connectivity to evaluate the effectiveness of the generated T in detecting these defects. Two scenarios can occur based on the problem description presented in Section 3. First, a defect is present at the border node of an LCZ; this defect is activated when the border node is visited during a process flow in the SUT. Given the test coverage criteria defined in Section 5, these defects were detected via LNCT. Thus, adding such defects to the evaluation was not considered a necessary step.

Second, a defect can be more complex and simulated as a pair (n_{out}, n_{back}), where $n_{out} \in N$ denotes a node of an SUT process model G in which the network connectivity is disrupted, which affects the SUT, and the defect is activated and demonstrated if the process flow goes to a node $n_{back} \in N$. 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 pairs in each G was set as a random number in the interval ranging from $\frac{1}{3}|\mathcal{L}|$ to $\frac{2}{3}|\mathcal{L}|$, and both interval boundaries were rounded to the nearest integer. In every LCZ, where defect pairs are generated, their number was equal to a random number in the interval ranging from $\frac{1}{3}$ to $\frac{2}{3}$ of the total number of combinations of the reachable LCZ IN and LCZ OUT nodes of this LCZ, both of which are rounded to the nearest integer.

The overall properties of the SUT models employed in the experiments are summarized in Table 2, in which we present the minimal (MIN), maximal (MAX), average ($\overline{x}$), and median ($\tilde{x}$) values for the individual model properties, as discussed previously.

The computation of T was performed on a machine running the Windows 10 platform and Java version 15.0.1 with the following hardware configuration: Intel(R) Core(TM) i5-10210U CPU @ 1.60 GHz, clock frequency of 2.11 GHz, 16 GB RAM, with an SSD disk.

8.3. Evaluation results

We describe the results of the EPP, SPC, and ANT algorithms for the coverage criteria specified in Section 5. The overall results of the algorithm are presented in Section 8.3.1. As presented in Section 8.3.2, the algorithm is analyzed, which yields the optimal results for the individual SUT models. Finally, we analyzed the effectiveness with which test cases produced by the individual algorithms detected the simulated limited connectivity network defects that are present in the SUT models.

8.3.1. Properties of test sets produced by the compared algorithms

First, we compare the properties of T generated by the individual algorithms using the evaluation criteria $\mathcal{E}$ introduced in Table 1. The results are averaged for all 150 SUT models for the *AllBorderCombinations* test-coverage criterion summarized in Table 3 and are presented in Fig. 4. The results for the *EachBorderOnce* criterion are summarized in Table 4 and are further visualized in Fig. 6.

For the ***AllBorderCombinations*** coverage, the ANT algorithm exhibits a significantly lower average value of $|T|$ than the EPP and SPC algorithms, yielding similar results for this criterion. The average ANT $|T|$ is 50% smaller than the average $|T|$ of the EPP algorithm. The average $|T|$ of EPP is only 1.8% smaller than the average $|T|$ of the SPC (see Table 3 and Fig. 4). Furthermore, the ANT algorithm outperformed the EPP and SPC in terms of the average total length of test cases $l(T)$. The ANT's average $l(T)$ is 15% smaller than the EPP and 20% smaller than the SPC. Considering the criterion b_nodes that represents the average number of border nodes in T, the ANT algorithm yields the superior result, which is approximately 26% less than the EPP, and its result is approximately 1.5% smaller than that of the SPC algorithm. The results of the ANT algorithm for $|T|$, $l(T)$, and b_nodes are compensated by the longer test cases produced by this algorithm. Considering $\overline{|t|}$, the average value of the results rendered by the ANT algorithm is 85% higher than that of the SPC algorithm and 90% higher than that of the EPP algorithm.

Table 3 lists the average runtimes of the compared algorithms in milliseconds. For this criterion, EPP emerges as the fastest algorithm with an average runtime of 3.96 ms, which is followed by SPC with an average runtime of 8.44 ms. The ANT algorithm

[3] List of the projects is available in XLSX, CSV, Open Excel, and PDF formats at: http://still.felk.cvut.cz/lnct/.

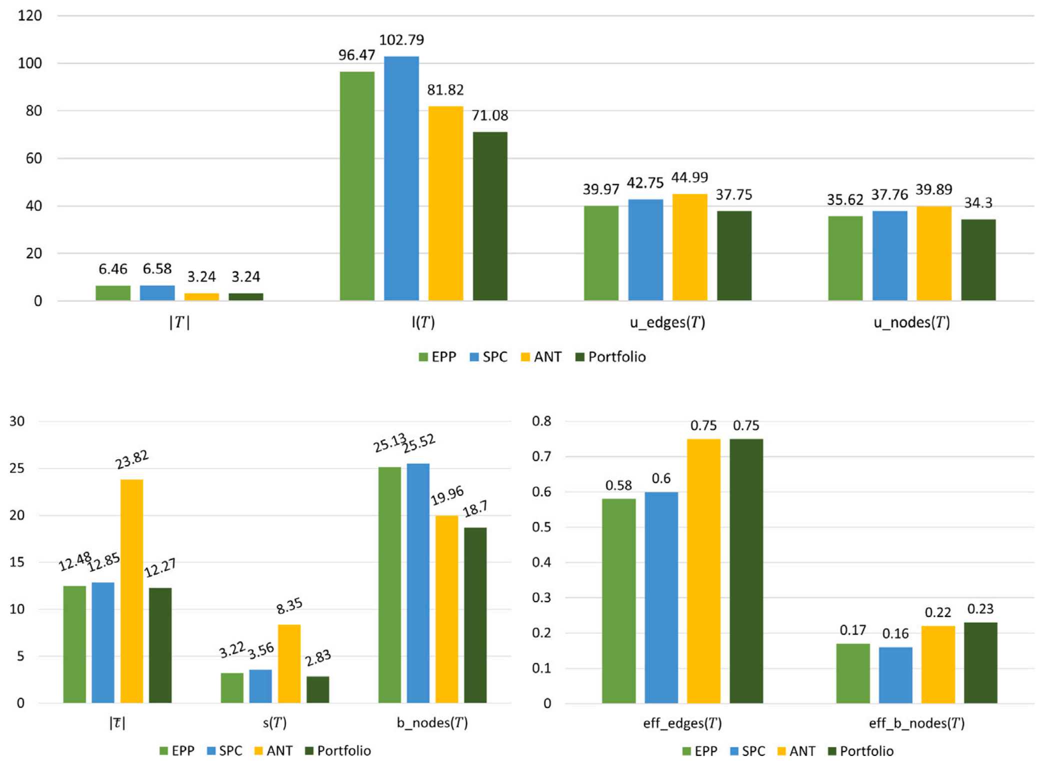


Fig. 4. Algorithm and portfolio strategy comparison for $AllBorderCombinations$ coverage through the evaluation criteria $\mathcal{E}$.

considered a considerably longer time to compute T, i.e., 190.05 ms on average. The longest run time measured during the experiments are 65.30 ms for EPP (for SUT model $id = 137$, $|N| = 221$, and $|E| = 377$, refer to Table 2), followed by 0.54 s for SPC (SUT model $id = 121$, $|N| = 83$, and $|E| = 144$) and by 6.22 s for ANT (SUT model $id = 148$, $|N| = 376$, and $|E| = 522$). For the largest SUT model ($|N| = 442$ and $|E| = 606$, $id = 150$ in Table 2), the runtime of the SPC algorithm is 13.90 ms, followed by those of the EPP algorithm with 51.88 ms and ANT algorithm with 1.39 s.

The results for $|T|$ are similar to those for *AllBorderCombinations* for the ***EachBorderOnce*** coverage criteria. The ANT algorithm possesses the smallest average size of the produced test set, which is 41% smaller than that of the average $|T|$ for the test sets produced by SPC, and 41% smaller than that of the average $|T|$ for the EPP algorithm (refer to Table 4 and Fig. 6). In contrast, the average $l(T)$ differs for *EachBorderOnce* coverage compared to *AllBorderCombinations* coverage. The average $l(T)$ is approximately identical for the ANT and SPC algorithms, with a difference of only approximately 0.5

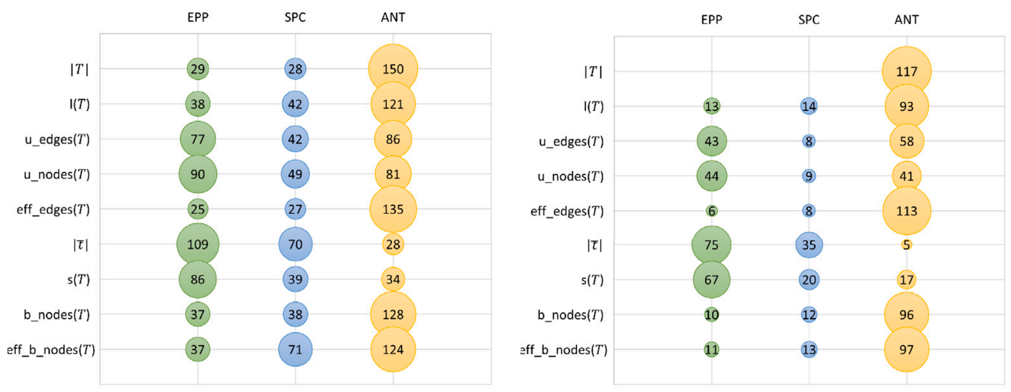
The average numbers of unique nodes $u\_nodes(T)$ and unique edges $u\_edges(T)$ contained in T for the *EachBorderOnce* coverage criteria are similar to those for the previous *AllBorderCombinations* coverage. The ANT algorithm visits the largest number of unique nodes and edges on average, whereas the EPP algorithm visits the smallest number of unique nodes and edges on average. The ANT possesses 11% more $u\_nodes(T)$ and 13% more $u\_edges(T)$ on average compared with the same criteria for the test sets produced by the EPP algorithm, and 9% more $u\_nodes(T)$ and 7% more $u\_edges(T)$ produced by the SPC algorithm.

The trend in the average value of $\overline{|t|}$ is similar for the *EachBorderOnce* criterion compared to the previous *AllBorderCombinations* coverage. The test sets produced by the ANT algorithm exhibit more than an 87% larger average $\overline{|t|}$ than those produced by the SPC algorithm. Moreover, they have a 97% larger average $\overline{|t|}$ than the EPP algorithm, which produced test sets with the shortest average length of test cases. For *EachBorderOnce*, the EPP algorithm outperformed the others in the length dispersion criterion $s(T)$ which is 15% lower than the average $s(T)$ of the SPC algorithm and 62% lower than the average $s(T)$ of the ANT algorithm.

The SPC and EPP algorithms yielded the same result equal to 0.19 for the average $eff\_b\_nodes(T)$ ratio, which is 17% less than the average $eff\_b\_nodes(T)$ of 0.23.

Considering the average runtime of the algorithms (refer to the last line of Table 4), the EPP emerges as the fastest algorithm with an average runtime of 3.27 ms, which is closely followed by the SPC with an average runtime of 4.45 ms. Similarly, as in the *AllBorderCombinations* coverage criterion, the ANT algorithm required the longest time to compute T, i.e., 135.40 ms on average. The longest runtime measured during the experiments for the *EachBorderOnce* test-coverage criterion is 44.84 ms for the EPP (for SUT model $id = 137$, $|N| = 221$, and $|E| = 377$, refer to Table 2), followed by 0.27 s for the SPC (SUT model $id = 121$, $|N| = 83$, and $|E| = 144$), and by 4.08 s for ANT (SUT model $id = 148$, $|N| = 376$, and $|E| = 522$).

The SUT models for which the EPP, SPC, and ANT algorithms required the longest time to compute T for the *AllBorderCombinations* and *EachBorderOnce* test-coverage criteria are the same. For the largest SUT model ($|N| = 442$ and $|E| = 606$, $id = 150$ in



(a) There can be more than one winning algorithm for each criteria; the results are sorted using the $\geq$ comparator.

(b) There can be only one winning algorithm for each criteria; the results are sorted using the $>$ comparator.

Fig. 5. Overall statistics of the test set selection strategy for $AllBorderCombinations$.

Table 2), the runtime of the SPC algorithm is 12.25 ms, which is followed by those of the EPP and ANT algorithms at 44.29 ms and 1.33 s, respectively.

8.3.2. *Algorithms that produced superior test sets for particular SUT models*

The averaged properties of T produced by the individual algorithms, as discussed in Section 8.3.1, yielded beneficial insights into the performance of the algorithms. However, it is only one of the possible viewpoints on the results. Moreover, we aimed to determine the algorithm showing the optimal result (considering a particular $\mathcal{E}$) for the individual SUT models.

First, the results for *AllBorderCombinations* coverage are presented in Fig. 5, and the results for *EachBorderOnce* coverage are illustrated in Fig. 7. In Figs. 5 and 7, the y-axis presents the individual test-set evaluation criteria introduced in Section 6 on the x-axis, which are individual algorithms. Each number at the intersection of the axes represents the number of cases in which a specific algorithm returns the optimal T considering specific evaluation criteria. This number is visually emphasized by the size of its surrounding bubble.

The difference between Figs. 5(a) and 5(b) lies in the comparison operator employed to select the winning algorithm. The $\geq$ comparator is utilized for the a part (left side) in these graphs. Therefore, if T_1 exhibits a better value for the evaluation criterion in the test sets T_1 and T_2 produced by Algorithms 1 and 2, then Algorithm 1 is considered the winner. Algorithms 1 and 2 are considered winners when the value of the evaluation criterion is identical for both T_1 and T_2. For the b part (right side) of the graph in Figs. 5 and 7, the $>$ comparator is utilized such that when in the test sets T_1 and T_2 produced by Algorithms 1 and 2, T_1 offers a better value for the evaluation criterion, and Algorithm 1 is considered the winner. No algorithm is considered a winner when the value of the evaluation criterion is identical for T_1 and T_2. The same difference applies to Figs. 7(b) and 7(a).

First, we characterized the performance of the algorithms using a $>$ comparator. Fig. 5(b) indicates that the ANT algorithm outperforms the others in several criteria for *AllBorderCombinations* coverage. This algorithm achieved the optimal result in 78% of the SUT models for $|T|$ in more than 75% of the models for $eff\_edges(T)$, which is approximately 65% of the models for $eff\_b\_nodes(T)$, 64% for $b\_nodes(T)$, and 62% for $l(T)$. However, the EPP algorithm achieved the optimal results in the case of 50% and approximately 45% of the models for the $\overline{|t|}$ and $s(T)$ criteria, respectively. Both the ANT and EPP algorithms returned similar results for the $u\_nodes(T)$ and $u\_edges(T)$ criteria. For *EachBorderOnce* coverage, Fig. 7(a) illustrates a scenario similar to the previous coverage criterion. The ANT algorithm outperformed the others in over 50% of the models for the $|T|$, $l(T)$, $eff\_edges(T)$, and $eff\_b\_nodes(T)$ criteria. For $b\_nodes(T)$, the results are not as satisfactory, i.e., the ANT algorithm clearly wins for only 41% of the models.

The EPP algorithm achieved the optimal value of $\overline{|t|}$ for 56% of the models and $s(T)$ for approximately 45% of the models. This scenario is similar to the case of the *AllBorderCombinations* criterion for the $u\_nodes(T)$ and $u\_edges(T)$ criteria, wherein the ANT and EPP algorithms won an equal number of cases.

The ANT algorithm achieved superior results for more than 80% of the cases considering $|T|$, $eff\_edges(T)$, $b\_nodes(T)$, $eff\_b\_nodes(T)$, and $l(T)$ for the *AllBorderCombinations* coverage (depicted in Fig. 5(a)) criteria when more than one winner could be allowed for the given criteria (using the $>$ comparator). None of the other algorithms performed optimally for any of the remaining criteria.

The results for the *EachBorderOnce* coverage criterion (Fig. 5(a)) cannot achieve a clear conclusion, i.e., the ANT algorithm wins in more than 80% of the cases only in the $|T|$ criteria.

Table 3
Average values of evaluation criteria over all G for the *AllBorderCombinations* coverage criterion.

Evaluation criterion	Algorithm			
	EPP	SPC	ANT	Portfolio
$\|T\|$	6.46	6.58	3.24	3.24
l(T)	96.47	102.79	81.82	71.08
u_edges(T)	39.97	42.75	44.99	37.75
u_nodes(T)	35.62	37.76	39.89	34.3
$\overline{\|t\|}$	12.48	12.85	23.82	12.27
s(T)	3.22	3.56	8.35	2.83
b_nodes(T)	25.13	25.52	19.96	18.7
eff_edges(T)	0.58	0.6	0.75	0.75
eff_b_nodes(T)	0.17	0.16	0.22	0.23
time [ms]	3.96	8.44	190.05	1.74

Table 4
Average values of evaluation criteria over all G for the *EachBorderOnce* coverage criterion.

Evaluation criterion	Algorithm			
	EPP	SPC	ANT	Portfolio
$\|T\|$	5.43	4.88	2.87	2.86
l(T)	75.8	69.3	68.89	58.29
u_edges(T)	38.25	39.7	43.22	36.43
u_nodes(T)	34.85	36.37	38.95	33.68
$\overline{\|t\|}$	11.97	12.61	23.53	11.8
s(T)	3.04	3.56	8.08	2.65
b_nodes(T)	19.35	17.38	16.53	14.9
eff_edges(T)	0.65	0.69	0.78	0.8
eff_b_nodes(T)	0.19	0.19	0.23	0.24
time [ms]	3.27	4.45	135.40	1.18

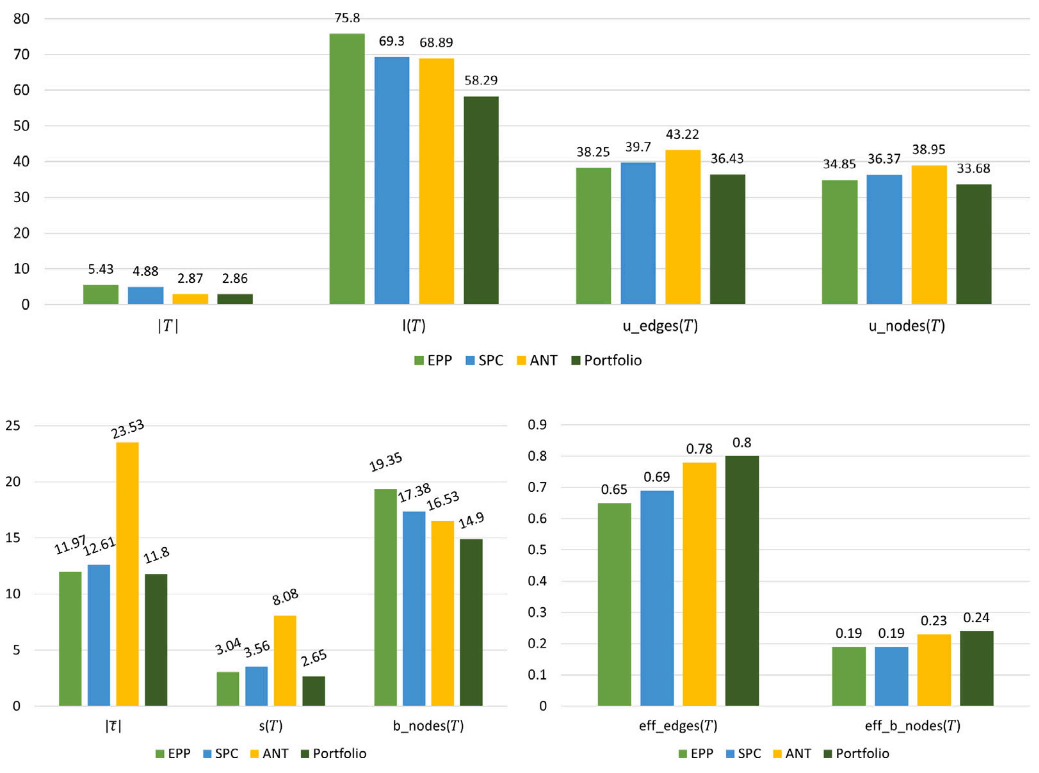
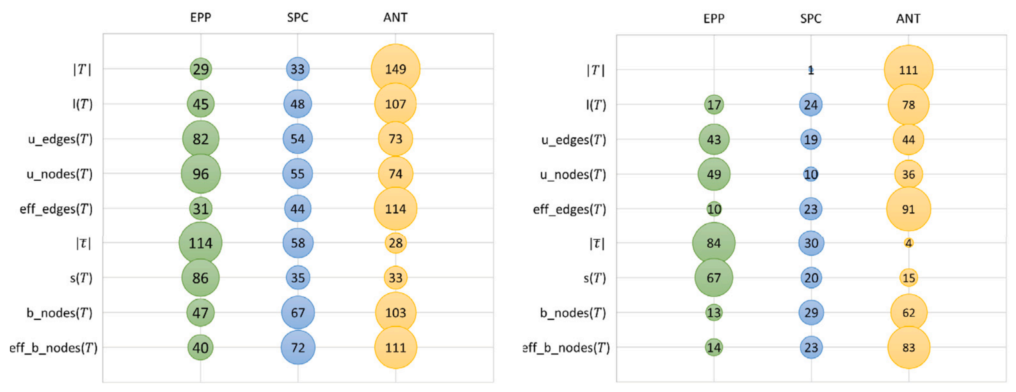


Fig. 6. Algorithm and portfolio strategy comparison for *EachBorderOnce* coverage through the evaluation criteria $\mathcal{E}$.



(a) There can be more then one winning algorithm for each criteria; the results are sorted using $\geq$ comparator.

(b) There can be only one winning algorithm for each criteria; the results are sorted using the $>$ comparator.

Fig. 7. Overall statistics of the test set selection strategy for *EachBorderOnce*.

8.3.3. Effectiveness of the detection of limited network connectivity defects in the SUT

A T value satisfying the *AllBorderCombinations* test-coverage criterion activates 100% of the defect pairs defined in G, resulting from the definition of *AllBorderCombinations*. Therefore, the results for *EachBorderOnce* warrant a more detailed analysis. In the experiments, T generated by the EPP detected 88% of the defect pairs on average for all SUT models. T generated by SPC detected 88%, and T generated by ANT detected 93% of these defects; these values are denoted as ψ.

These results must be considered in the context of the T size, i.e., $l(T)$. Herein, we consider $\Psi = \psi/\overline{l(T)}$ as an indicator, and we determine the effectiveness of T in the activation (visiting) of defect pairs inserted in all SUT models in the experiment; $\overline{l(T)}$ is the average of $l(T)$ for all SUT models. A higher Ψ indicates an improved effectiveness of the test sets generated by a particular algorithm. The Ψ values are 0.0116 for EPP, 0.0127 for SPC, and 0.0135 for ANT.

9. Portfolio strategy

This project utilizes two levels of test coverage criteria C owing to the problem complexity, variability in G topology, and principle of individual algorithms that can generate T. For a variety of test set evaluation criteria $\mathcal{E}$, a portfolio strategy must be used to generate the most close-to-optimum T for a general G.

As defined in Algorithm 13, the proposed portfolio strategy accepts G, $threshold$, C, and $\mathcal{E}$ as inputs, generating T as the output. This strategy computes the individual T for G and $threshold$ using the SPC, ANT, and EPP algorithms. Subsequently, it determines the optimal T by considering $\mathcal{E}$. In particular, the SPC, ANT, and EPP algorithms can be executed concurrently to optimize the computation runtime.

Algorithm 13: $portfolio(G, threshold, C, \mathcal{E})$: Compute T for G and $threshold$ by SPC, ANT and EPP algorithms and determine the best T by $\mathcal{E}$

```
Input  : SUT model $G$, $threshold$, coverage criterion $C$ and test set optimality criterion $\mathcal{E}$
Output: set of test cases $T$
1 $T_{SPC} \leftarrow \emptyset$, $T_{ANT} \leftarrow \emptyset$, $T_{EPP} \leftarrow \emptyset$
2 $T_{SPC} \leftarrow SPC(G, threshold, C)$
3 $T_{ANT} \leftarrow ANT(G, threshold, C)$
4 $T_{EPP} \leftarrow EPP(G, threshold, C)$
5 $T \leftarrow$ a test set from $\{T_{SPC}, T_{ANT}, T_{EPP}\}$ having the best value of given $\mathcal{E}$
6 return $T$
```

The results of the portfolio strategy are presented in Fig. 4 for the *AllBorderCombinations* and in Fig. 6 for the *EachBorderOnce* criteria. The average $|T|$ for the *AllBorderCombinations* coverage is the identical to that for the ANT algorithm in terms of the results of the portfolio strategy, which is approximately 50% smaller than the average $|T|$ of the EPP and 51% smaller than the average $|T|$ of SPC. The portfolio strategy returned T with 13% smaller $l(T)$ (the total length of the test cases) than the ANT algorithm, 26% smaller than the EPP algorithm, and approximately 31% smaller than the SPC algorithm for the total length of the test cases.

Considering $u\_edges(T)$, the portfolio strategy generates a value approximately 6% smaller than that of the EPP algorithm, which is approximately 12% smaller than that of the SPC algorithm and more than 16% smaller than the results of this criterion for T

generated by the ANT algorithm. For the $u\_nodes(T)$ criterion, only a 4% difference is observed between the results of the portfolio strategy and EPP algorithm. Otherwise, the portfolio strategy has 9% smaller $u\_nodes(T)$ than the SPC algorithm, and 14% smaller nodes than the ANT algorithm. Considering the average number of border nodes b_nodes in T, the portfolio strategy yields a value greater than 6%, which is smaller than the ANT algorithm, i.e., approximately 26% less than the EPP algorithm and 27% smaller than the SPC algorithm.

The average $|T|$ value is nearly the same for both the portfolio strategy and the ANT algorithm for ***EachBorderOnce*** coverage. However, the portfolio strategy returns T with more than 41% lower $|T|$ than the SPC algorithm and more than 47% lower values than the EPP algorithm. The average value for the results generated using the portfolio strategy is more than 15% lower than that of the ANT algorithm for the $l(T)$ criterion, which is nearly 16% lower than that of the SPC algorithm and more than 23% lower than that of the EPP algorithm. Considering the $u\_edges(T)$ criterion, the portfolio strategy achieves a nearly 5% lower value of this criterion compared to the EPP algorithm, which is more than 8% smaller than the SPC algorithm, and nearly 16% smaller than the ANT algorithm. For the $u\_nodes(T)$ criterion, the portfolio strategy produces T with a value more than 3% lower than that of the EPP algorithm; this is more than 7% smaller than that of the SPC algorithm, and nearly 14% smaller than that of the ANT algorithm.

The portfolio strategy selects the best T for the individual results provided by the SPC, ANT, and EPP algorithms. Therefore, it is not included in the overall statistics presented in Figs. 5 and 7.

For *AllBorderCombinations*, the portfolio strategy produced T that detected 100% of the defect pairs defined in G. For *EachBorderOnce*, on average, the portfolio strategy produced T which activated 95.6% of the defect pairs defined in individual G in the experiments. For the portfolio strategy, $\overline{l(T)}$ is 58.29, and Ψ is 0.0164, which is 21%, 29

10. Discussion

The main finding from the performed experiments is that, despite the ANT yielding the best results for the number of criteria and most SUT models (Figs. 5 and 7), it is not a clear winner that would yield the best T considered for all SUT models. This scenario occurs only once for $|T|$ and *AllBorderCombinations*.

Fig. 5(a) shows that even the SPC algorithm for some criteria (e.g., $u\_nodes(T)$, $eff\_b\_nodes(T)$, or $\overline{|t|}$) returns results comparable to those generated by the other two algorithms. Furthermore, the results generated by this algorithm seem to be better distributed, especially when we look at those for the *EachBorderOnce* coverage depicted in Fig. 7(b). The SPC algorithms return results almost as good as the other two algorithms for all measured criteria, which range from the $|T|$ criteria, which have the same size as the winner in 22% of the cases, to the $eff\_b\_nodes(T)$ criteria, which return the same result as the winner in 48% of the cases. Thus, using the three algorithms examined in this study in the portfolio strategy presented in Section 9 is the optimal option for constructing the best T for G.

An analysis of the differences between the test sets generated using the portfolio strategy and the algorithm achieving the optimal result for the compared criteria for $|T|$ indicates that $u\_nodes(T)$, $\overline{|t|}$, $eff\_edges(T)$, and $eff\_b\_nodes(T)$ are close (less than 5%). Thus, these results conclusively indicate that the winning algorithms are adequately designed and achieve optimal results for these criteria. In contrast, the most significant difference between the results of the portfolio strategy and winning algorithms for both coverage criteria is 15.4%, which corresponds to the $l(T)$ criterion for T generated by the ANT algorithm under *EachBorderOnce* coverage.

The ANT algorithm generates a lower number ($|T|$) of longer test cases but with fewer steps in total ($l(T)$), as revealed by analyzing the functionality of individual algorithms. This result corresponds well with the algorithm principle: ants attempt to visit more LCZs in a single test case. Shorter test cases are produced in the case of SPC; however, they may involve more steps in total. This is a result of this algorithm's “greedy” nature when traversing G. In contrast to ANT, SPC does not consider the proximity of the individual LCZs. Moreover, the performance of EPP is slightly superior than that of the SPC optimization part presented in its set-covering algorithm and by splitting the super test requirement to T (refer to Algorithm 11).

The maximum runtime of the ANT algorithm during the entire experiment (6.22 s) was considered acceptable. This runtime was for the SUT model with 376 nodes and 522 edges, which is a rare scenario in industrial praxis. We expect that models with more than 100 nodes are less common. An analysis of the highest runtime versus runtime for the largest SUT model indicates that not only does the G size in terms of $|N|$ and $|E|$ play a role, but the graph topology combined with the presence of LCZs is more critical. Interestingly, the SUT models for which the algorithms required the longest time to compute T were identical for all algorithms and both test coverage criteria. During runtimes, the principle of the algorithms and number of cycles present in G play a significant role.

Simulated defect pairs were defined in the SUT models for the effectiveness of T in detecting limited network connectivity. The ANT algorithm achieved the best results for the Ψ criterion in *EachBorderOnce*. Furthermore, the portfolio strategy outperformed ANT by 21% in terms of Ψ. A simple defect type of limited network connectivity, activated by visiting either the LCZ IN or LCZ OUT node, will be activated by the test set for both test coverage criteria proposed in this study. This scenario was also observed with the *AllBorderCombinations* test-coverage criterion for defect pairs. The presented simulation yields initial insight. However, real defects in an SUT must be investigated in a forthcoming study to evaluate this type of effectiveness more accurately.

11. Threats to validity

Several concerns can be raised regarding the validity of these experiments. First, the experimental data could be biased owing to the small sample size. We executed all the algorithms in the experiment on 150 various SUT models to mitigate this potential problem, yielding a sufficient variety of situations in which the algorithms can exercise to draw conclusions. The SUT models are elucidated in Section 8.2.

Second, the relevance of used SUT models to real-life cases of the workflows in real-time IoT systems is an issue of concern. The analysis of 50 or more different real-time IoT systems is not feasible because of the confidential nature of internal structures employed in industrial IoT systems. A balance between the number of SUT models used in the experiments and their similarity to real-world cases should be determined to solve this problem. We utilized the process described in Section 8.2 to ensure that the SUT models approximated the real systems. The initial set of 22 workflows was based on real systems, and an extended set comprised the remaining 98 models. Selected details of the workflows were applied to maintain the topology of the workflows in the remaining SUT models. Only 30 SUT models used in the experiments were generated using specialized software; these models were utilized to ensure heterogeneous topology of the models.

Another concern was identified regarding the EPP algorithm used for comparison with the discussed algorithms. Among the available algorithms suitable for serving as the baseline, the algorithm proposed by Li et al. [9] is the most recent. Furthermore, a possible bias may have resulted from the incorrect implementation of the EPP core, the set-covering algorithm [9]. This issue was circumvented by employing the original implementation of the algorithm reported by Offutt, Ammann, Li, Xu, and Deng.[4]

The algorithms were compared using the evaluation criteria related to T (see Section 6). The defect detection power, i.e., the number of defects in an SUT discovered by a particular T, was analyzed by simulating the defects concerning limited network connectivity in the SUT model. An initial insight into this effectiveness was obtained; however, further experiments are required to obtain more accurate data. Mutation testing [50,51] or defect-injection experiments [52] are required for several SUT instances with different sets of inserted defects to obtain more data. However, in principle, defects concerning limited network connectivity are difficult to simulate using code mutations or defect insertion. Therefore, an appropriate experiment must be designed to maintain the objectivity of the experiment.

12. Conclusion

We proposed a novel technique for path-based testing of the behavior of an IoT system wherein the functionality of its components was influenced by limited network connectivity. For example, when the network connectivity outage or bandwidth was limited, it could affect the functionality of a system component. This technique was based on modeling an SUT process or workflow aspect, and could capture the network outage probability in the model. The test cases included a flow of SUT functions, such that they were sequenced according to the test coverage criterion when the network connectivity was disrupted and restored. In contrast, the number of steps in other parts of the SUT model was minimal. Using this principle, the testing costs were minimized, and they were compared to other ad hoc approaches that could be considered for solving the problem. No previous path-based testing algorithms could be utilized directly because the direct support to follow a particular node (edge) after another node (edge) visited in the test case was not present in these algorithms.

The major contributions of this paper include: (1) the definition of LNCT in an IoT system from the viewpoint of system functionality, (2) two algorithms (a new SPC using the principle of the shortest path composition and ANT, a novel application of the ACO principle to solve the problem discussed), (3) a detailed evaluation of these algorithms in comparison to EPP as an alternative based on the established prime-path composition approach, and (4) a portfolio strategy that employed all of these algorithms to obtain the superior T.

The ANT algorithm achieved the best results for most of the 150 SUT models employed in the experiments, considering the evaluation criteria of the defined test set, total number of test steps, average length of test cases, and other parameters (Table 1). However, SPC and EPP yielded better results than ANT in certain SUT instances.

We could consider $l(t)$ as the main criterion that approximated the potential effort required to execute the test cases for highlighting key findings. For the *AllBorderCombinations* test-coverage criterion and $>$ comparator, the ANT algorithm clearly achieved the optimal test set for 93 SUT models, whereas the SPC and EPP algorithms achieved the optimal test set for 14 and 13 models, respectively. Observably, the ANT algorithm computed the best test set for the 78 SUT models concerning the *EachBorderOnce* and $>$ comparator; the SPC and EPP algorithms computed the superior test set for 24 and 17 models out of 150 total models. Therefore, although ANT was considered the best option (owing to the individual options of $\mathcal{E}$ used in this study) to guarantee the optimal T, all the algorithms examined must be combined in the portfolio strategy presented in this paper. Furthermore, this strategy achieved the best results, considering the potential of the produced test sets in detecting simulated defects concerning limited connectivity. In future research, we shall enhance the portfolio strategy by including other algorithms to compute T. The core principle of these algorithms will differ primarily from SPC, ANT, and EPP, aiming to improve the probability that the added algorithm will contribute to the computation of T as optimally as possible for any given topology of the SUT model.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Miroslav Bures and Matej Klima has patent Method of Testing of IoT System Behavior in Case of Limited Network Connection issued to Czech Technical University in Prague.

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

Data availability

Data will be made available on request.

Acknowledgments

The high-level principle of a technique including some initial algorithm that is not included in this study is also described in a US patent application US2021311857A1 by our team. The details are available at: https://worldwide.espacenet.com/patent/search/family/074221195/publication/US2021311857A1?q=pn%3DUS2021311857A1

The project is supported by CTU in Prague internal grant SGS20/177/ OHK3/3T/13 *Algorithms and solutions for automated generation of test scenarios for software and IoT systems*.

The authors acknowledge the support of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 *Research Center for Informatics*.

Bestoun S. Ahmed has been supported by the Knowledge Foundation of Sweden (KKS) through the Synergi Project *AIDA - A Holistic AI-driven Networking and Processing Framework for Industrial IoT* (Rek:20200067).

References

[1] S. Li, L. Da Xu, S. Zhao, The internet of things: a survey, Inf. Syst. Front. 17 (2) (2015) 243–259.
[2] T. Qiu, N. Chen, K. Li, M. Atiquzzaman, W. Zhao, How can heterogeneous Internet of Things build our future: A survey, IEEE Commun. Surv. Tutor. 20 (3) (2018) 2011–2027.
[3] B.S. Ahmed, M. Bures, K. Frajtak, T. Cerny, Aspects of quality in internet of things (IoT) solutions: A systematic mapping study, IEEE Access 7 (2019) 13758–13780.
[4] M. Bures, M. Klima, V. Rechtberger, B.S. Ahmed, H. Hindy, 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, Springer, Cham, 2021, pp. 546–556.
[5] D.E. Kouicem, A. Bouabdallah, H. Lakhlef, Internet of things security: A top-down survey, Comput. Netw. 141 (2018) 199–221.
[6] B.B. Gupta, M. Quamara, An overview of Internet of Things (IoT): Architectural aspects, challenges, and protocols, Concurr. Comput.: Pract. Exper. 32 (21) (2020) e4946.
[7] Y. Ding, M. Jin, S. Li, D. Feng, Smart logistics based on the internet of things technology: an overview, Int. J. Logist. Res. Appl. 24 (4) (2021) 323–345.
[8] M. Bures, T. Cerny, 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.
[9] N. Li, F. Li, 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.
[10] P. Ammann, J. Offutt, Introduction to Software Testing, Cambridge University Press, 2016.
[11] V. Arora, R. Bhatia, M. Singh, Synthesizing test scenarios in uml activity diagram using a bio-inspired approach, Comput. Lang. Syst. Struct. 50 (2017) 1–19.
[12] M. Bures, B.S. Ahmed, Employment of multiple algorithms for optimal path-based test selection strategy, Inf. Softw. Technol. 114 (2019) 21–36.
[13] M. Bures, B.S. Ahmed, K.Z. Zamli, Prioritized process test: An alternative to current process testing strategies, Int. J. Softw. Eng. Knowl. Eng. 29 (07) (2019) 997–1028.
[14] S. Anand, E.K. Burke, T.Y. Chen, J. Clark, M.B. Cohen, W. Grieskamp, M. Harman, M.J. Harrold, P. Mcminn, A. Bertolino, et al., An orchestrated survey of methodologies for automated software test case generation, J. Syst. Softw. 86 (8) (2013) 1978–2001.
[15] T. Su, K. Wu, W. Miao, G. Pu, J. He, Y. Chen, Z. Su, A survey on data-flow testing, ACM Comput. Surv. 50 (1) (2017).
[16] A. Khamis, R. Bahgat, R. Abdelaziz, Automatic test data generation using data flow information, 2000.
[17] H.M. Salman, A.K.M. Al-Qurabat, A.A.R. Finjan, Bigradient neural network-based quantum particle swarm optimization for blind source separation, IAES Int. J. Artif. Intell. (IJ-AI) 10 (2) (2021) 355.
[18] D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview, Soft Comput. 22 (2) (2018) 387–408.
[19] A. Windisch, S. Wappler, J. Wegener, Applying particle swarm optimization to software testing, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 1121–1128.
[20] P.R. Srivastava, K. Baby, 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.
[21] P.R. Srivastava, N. Jose, S. Barade, D. Ghosh, Optimized test sequence generation from usage models using Ant colony optimization, Int. J. Softw. Eng. Appl. 2 (2) (2010) 14–28.
[22] F. Sayyari, 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.
[23] S. Muthiah, R. Venkatasubramanian, The internet of things : QA unleashed, 2015.
[24] G. Murad, A. Badarneh, A. Qusef, F. Almasalha, Software testing techniques in IoT, in: 2018 8th International Conference on Computer Science and Information Technology, CSIT, 2018, pp. 17–21.
[25] M. Sirshar, K. Naeem, M. Khan, T. Akbar, Software quality assurance testing methodologies in IoT, 2019.
[26] J. Esquiagola, L.C. de Paula Costa, P. Calcina, G. Fedrecheski, M. Zuffo, Performance testing of an internet of things platform, in: IoTBDS, 2017, pp. 309–314.
[27] H. Rudeš, I.N. Kosović, T. Perković, 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.
[28] G. White, V. Nallur, S. Clarke, Quality of service approaches in IoT: A systematic mapping, J. Syst. Softw. 132 (2017) 186–203.
[29] S.H. Alsamhi, F.A. Almalki, H. Al-Dois, S. Ben Othman, J. Hassan, A. Hawbani, R. Sahal, B. Lee, H. Saleh, Machine learning for smart environments in B5G networks: connectivity and QoS, Comput. Intell. Neurosci. 2021 (2021).
[30] A.P. Matz, J.-A. Fernandez-Prieto, U. Birkel, et al., A systematic analysis of narrowband IoT quality of service, Sensors 20 (6) (2020) 1636.
[31] A.K.M. Al-Qurabat, Z.A. Mohammed, Z.J. Hussein, Data traffic management based on compression and MDL techniques for smart agriculture in IoT, Wirel. Pers. Commun. 120 (3) (2021) 2227–2258.
[32] A. Al-Qurabat, A lightweight huffman-based differential encoding lossless compression technique in IoT for smart agriculture, IJCDS J. 11 (2022) 117–127.
[33] I. Dakhil Idan Saeedi, A. Al-Qurabat, Perceptually important points-based data aggregation method for wireless sensor networks, Baghdad Sci. J. 19 (2022) 875–886.

[34] I.D.I. Saeedi, A.K.M. Al-Qurabat, An energy-saving data aggregation method for wireless sensor networks based on the extraction of extrema points, AIP Conf. Proc. 2398 (1) (2022) 050004.
[35] A.K.M. Al-Qurabat, H.M. Salman, A.A.R. Finjan, Important extrema points extraction-based data aggregation approach for elongating the WSN lifetime, Int. J. Comput. Appl. Technol. 68 (4) (2022) 357–368.
[36] A.K.M. Al-Qurabat, S.A. Abdulzahra, A.K. Idrees, Two-level energy-efficient data reduction strategies based on SAX-LZW and hierarchical clustering for minimizing the huge data conveyed on the internet of things networks, J. Supercomput. 78 (16) (2022) 17844–17890.
[37] A. Abdulzahra, A. Al-Qurabat, A clustering approach based on fuzzy C-means in wireless sensor networks for IoT applications, Karbala Int. J. Mod. Sci. 8 (2022) 579–595.
[38] H. Kim, A. Ahmad, J. Hwang, H. Baqa, F. Le Gall, M.A. Reina Ortega, J. Song, IoT-TaaS: Towards a prospective IoT testing framework, IEEE Access 6 (2018) 15480–15493.
[39] A. Rayes, S. Salam, Internet of things (IoT) overview, in: Internet of Things from Hype to Reality, Springer, 2019, pp. 1–35.
[40] A.K. M. Al-Qurabat, S. Abdulhussein Abdulzahra, An overview of periodic wireless sensor networks to the internet of things, IOP Conf. Ser. Mater. Sci. Eng. 928 (3) (2020) 032055.
[41] A.K. Gomez, 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.
[42] M. Noura, M. Atiquzzaman, M. Gaedke, Interoperability in internet of things: Taxonomies and open challenges, Mob. Netw. Appl. 24 (3) (2018) 796–809.
[43] M. Klima, M. Bures, A testing tool for IoT systems operating with limited network connectivity, in: A. Rocha, H. Adeli, G. Dzemyda, F. Moreira, A.M. Ramalho Correia (Eds.), Trends and Applications in Information Systems and Technologies, Springer International Publishing, Cham, 2021, pp. 570–576.
[44] M. Bures, T. Cerny, 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.
[45] A. Dwarakanath, 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.
[46] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. B 26 (1) (1996) 29–41.
[47] M. Dorigo, 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, pp. 1470–1477.
[48] M. Bures, Pctgen: Automated generation of test cases for application workflows, in: New Contributions in Information Systems and Technologies, Springer, 2015, pp. 789–794.
[49] V. Aravindan, D. James, Smart homes using Internet of Things, Int. Res. J. Eng. Technol. (2017) 1725–1729.
[50] P. Reales, M. Polo, J.L. Fernandez-Aleman, A. Toval, M. Piattini, Mutation testing, IEEE Softw. 31 (3) (2014) 30–35.
[51] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (5) (2010) 649–678.
[52] M. Bures, P. Herout, 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, 2020, pp. 442–447.