Uncertainty-wise Test Case Generation and Minimization for Cyber-Physical Systems

Man Zhang[1], Shaukat Ali[2], Tao Yue[2,3]
[1]Kristiania University College, Oslo, Norway
[2]Simula Research Laboratory, Oslo, Norway
[3]Nanjing University of Aeronautics and Astronautics
man.zhang@kristiania.no, {shaukat, tao}@simula.no

Abstract
Cyber-Physical Systems (CPSs) typically operate in highly indeterminate environmental conditions, which require the development of testing methods that must explicitly consider uncertainty in test design, test generation, and test optimization. Towards this direction, we propose a set of uncertainty-wise test case generation and test case minimization strategies that rely on test ready models explicitly specifying subjective uncertainty. We propose two test case generation strategies and four test case minimization strategies based on the Uncertainty Theory and multi-objective search. These strategies include a novel methodology for designing and introducing indeterminacy sources in the environment during test execution and a novel set of uncertainty-wise test verdicts. We performed an extensive empirical study to select the best algorithm out of eight commonly used multi-objective search algorithms, for each of the four minimization strategies, with five use cases of two industrial CPS case studies. The minimized set of test cases obtained with the best algorithm for each minimization strategy were executed on the two real CPSs. The results showed that our best test strategy managed to observe 51% more uncertainties due to unknown indeterminate behaviors of the physical environments of the CPSs as compared to the other test strategies. Also, the same test strategy managed to observe 118% more unknown uncertainties as compared to the unique number of known uncertainties.

Keywords
Uncertainty, Cyber-Physical System, Test Case Generation and Minimization, Multi-Objective Search.

1 Introduction

Cyber-Physical Systems (CPSs) are destined to face uncertainty in their operation due to close interactions with their physical environment [1]. Thus, classical testing methods (e.g., regression testing [2], conformance testing [3, 4]) must be adapted to consider uncertainty explicitly. There exist a few methods in the literature that explicitly take uncertainty into account such as considering uncertainty in test data generation [5] and testing distributed real-time systems in the presence of time uncertainty [6]. However, it lacks a systematic approach for considering the uncertainty aspect throughout test design, test generation, and test execution.

There is potentially the vast number of scenarios and environmental conditions in which a CPS can be tested. Uncertainty exponentially increases the number of such scenarios given that uncertain situations in the environment can happen in parallel to the CPS behavior. Even assuming, only known uncertainties, it is impossible to write tests for all the scenarios and uncertainties manually. Thus, there is a need for a solution to specify such scenarios and uncertainties at a higher level of abstraction, while leaving the low-level coding tasks to the software tools. Thus, the first scientific challenge is *how to systematically and automatically generate cost-effective tests explicitly considering uncertainties*. In addition, to test whether a CPS is capable of handling uncertainties properly, it is prerequisite to enable the occurrence of uncertainties during testing. However, the occurrence of uncertainty is *uncertain* by nature. This poses the second challenge: *how to manipulate* (e.g., increase the chance of) *the occurrence of uncertainties during testing*. It is also common that uncertainties of CPSs and sources of these uncertainties (e.g., unpredictable operation environment) are not fully known, i.e., *unknown* uncertainty. So, discovering *unknowns* (i.e., making unknowns known) as much as possible before CPSs' deployments via testing is therefore important. Hence the third scientific challenge is *how to identify unknown uncertainties and these sources* via testing.

To address these challenges, we proposed an uncertainty-wise testing framework, named as *UncerTest*, by considering the uncertainty aspect in test generation, test execution environment, and test verdicts. The key contributions of UncerTest are summarized in Table 1, along with its objectives and corresponding challenges. In principle, UncerTest is a model-based testing (MBT) framework, as its test ready models require explicitly specifying *subjective* uncertainties, which are defined as "lack of knowledge" [7, 8] about the expected behavior of a CPS in the presence of uncertainty in its operating environment. Such models need to be developed with our previous work: Uncertainty Modeling Framework (UncerTum) [9, 10], and are specifically named as *Belief Test Ready Model*s (BMs)[1] in the rest of the paper.

We propose two test generation strategies in UncerTest: 1) the *All Simple Belief Paths* coverage (*ASiBP*): a set

[1] BMs have two types of UML diagrams: 1) *Belief Class Diagrams* (BCDs) capturing testing interfaces (e.g., observable states and operations) and 2) *Belief State Machines* (BSMs) modeling the expected behavior of a CPS with *subjective* uncertainty captured.

of all simple paths (no loops) in a *Belief State Machine*, each of which contains unique states and transitions; and 2) the *All Specified Length Belief Paths* coverage (*ASlBP*): a set of all paths in a *Belief State Machine*, the maximum length of each of which can be set to any positive number. Each path is an abstract test case. For each abstract test case, UncerTest automatically calculates uncertainty related properties, e.g., uncertainty measurements using an applicable mathematical model (uncertainty measurement calculation).

Followed by the test generation, the *Uncertainty-wise Test Minimization* approach of UncerTest should be applied if the number of automatically generated abstract test cases is large, which is often true for any non-trivial CPS, and it is practically impossible to execute all of them. The test case minimization strategies of UncerTest are formulated as multi-objective search problems with the aim of balancing cost, effectiveness, and uncertainty related objectives. Therefore, we opted for applying multi-objective search algorithms (e.g., NSGA-II) for achieving the minimization goal. More specifically, we proposed four multi-objective search problems, which aim to minimize the number of test cases (to reduce the execution cost) and maximize the transition coverage (to maintain the effectiveness of coverage of a test ready model). However, each problem attempts to achieve four different uncertainty related objectives: 1) maximizing the average *number of uncertainties* covered by the selected test cases, which aims to test more defined uncertainties; 2) maximizing the average *percentage of uncertainty space* covered by the selected test cases, which aims to test more uncertainties from the different uncertainty space; 3) maximizing the average *uncertainty measurement* of the selected test cases, which aims to test paths with high confidence; and 4) maximizing the average *percentage of unique uncertainties* covered by selected test cases, which aims to test more different uncertainties.

A minimized set of abstract test cases is then converted into executable test cases. To raise the chance of uncertainty occurrences, the invocation of source(s) of the uncertainty (i.e., indeterminacy sources) are encoded in the executable test cases, which varies for the four test case minimization strategies. To enable the observation of uncertainty occurrences, an uncertainty occurrence evaluation is employed in the executable test cases, which is an implementation of our newly proposed uncertainty-wise test verdicts.

We evaluated UncerTest with two industrial case studies: GeoSports (GS) [11] (with one use case) and Automotive Warehouse (AW) [12] (with four use cases). Results showed that UncerTest managed to generate test cases, minimize test cases, enable sources of uncertainty in the test execution environment, and perform assertions and generate uncertainty-wise test verdicts. To assess the cost-effectiveness of the test cases obtained by different strategies (combining generation and minimization) of UncerTest, we performed an empirical evaluation using the two case studies. Regarding the comparison across the test strategies to discover uncertainties in the behaviors of CPSs, our best strategy managed to discover 51% more uncertainties as compared to the rest of the test strategies due to unknown indeterminacy sources in the physical environments of the two industrial case studies. Also, the same test strategy observed 118% more unknown uncertainties due to unknown indeterminate behaviors of the physical environments as compared to the already known uncertainties.

Table 1. Challenges, Objectives, and Contributions

	Scientific Challenge	Objective	Contribution	
C1	*How to systematically and automatically generate cost-effective tests by explicitly considering uncertainties?*	Develop automated uncertainty-wise test case generation with MBT, and cost-effective test minimization with search	**Test generation:** Two automated test generation strategies to generate test cases with uncertainties and calculate uncertainty measurements. **Test minimization:** Four strategies to obtain a cost-effective set of test cases from the generated ones by considering four different dimensions of uncertainty.	**Tool Support**
C2	*How to manipulate the occurrence of uncertainties during testing*?	With MBT, allow specification of known possible sources of uncertainties in the models, and introduce the sources during test execution.	**Introduction of sources of uncertainties**: Extended *UncerTum* for modeling sources of uncertainty (i.e., indeterminacy sources) with recommendations; defining strategies to introduce the sources of uncertainty regarding which sources to introduce, where and how to introduce them; and enabling the introduction of the sources according to the specified strategies by executable test case generation	
C3	*How to identify unknown uncertainties and their sources*?	Devise uncertainty-wise test verdicts to discover unknowns by checking occurrences of known uncertainties.	**Test verdict:** Defining test verdicts by evaluating an occurrence of a specified behavior of a CPS together with the occurrence of uncertainties and their sources. Such test results can be used to identify *unknown* uncertainties and examine relationships between uncertainties and their sources.	

A list of abbreviations in the paper is shown in Appendix A along with descriptions. The rest of the paper is organized as follows. In Section 2, we briefly summarize UncerTum [10] and the Uncertainty Theory. In Section 3, we present a running example to illustrate the proposed approach throughout the paper. The overview of the proposed approach is presented Section 4, followed by abstract test case generation (Section 5), uncertainty-wise test minimization (Section 6), executable test case generation (Section 7) and uncertainty-wise test verdict (Section 8). The evaluation is discussed in Section 9, and the tool implementation is described in Section 10. We

introduced related work in Section 11 and concluded the paper in Section 12.

2 Background

Section 2.1 presents our previous work Uncertainty Modeling Framework (UncerTum) used for developing belief test ready models. Section 2.2 introduces two mathematical models for computing uncertainty measurements.

2.1 Uncertainty Modeling Framework (UncerTum)

UncerTum [9, 10] was proposed to develop test ready models for enabling MBT of CPSs in the presence of environmental uncertainty. UncerTum is equipped with specialized modeling notations (named as the UML Uncertainty Profile (UUP)) for specifying uncertainties. UUP is the core of UncerTum and UUP implements a conceptual model, named as U-Model [13]. U-Model was developed to understand uncertainties in CPSs by defining, characterizing and classifying uncertainties and associated concepts (e.g., *Belief*, *BeliefStatement*, *IndeterminacySource*, *Measure,* and *Measurement*), and their relationships at a conceptual level.

UncerTum additionally defines four sets of UML model libraries: *Pattern*, *Time*, *Measure*, and *Risk* libraries, by extending the existing UML profile: Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [14]. The purpose of defining these libraries is to ease the development of test ready models with uncertainty.

In summary, key notations used in UncerTum are standard UML state machines and class diagrams with UUP stereotypes and the model libraries applied. Such diagrams are referred as BMs[1]. More details about UncerTum can be found in our previous work: [9, 10].

2.2 Mathematical model for uncertainty measurements

In UncerTest, how to compute uncertainty measurements depends on which mathematical tool is applied. Regarding obtaining uncertainty measurements, *Probability Theory* is commonly [15] used for measuring uncertainty as frequencies based on samples from long-run experiments [15, 16], e.g., random experiments with repeated trials. However, an estimated value can be regarded as close enough to the long-run frequency only if the sample size is large enough. This is a fundamental prerequisite of applying *Probability Theory*. However, at the initial stage of testing (e.g., test design for enabling MBT), such large samples are often unavailable [16]. Therefore, we selected *Uncertainty Theory* (i.e., *"a branch of axiomatic mathematics for modeling belief degrees"*) to handle the situation of not having sufficient samples and therefore measuring the *belief degree* of an uncertainty subjectively by domain experts [16, 17], instead of using *frequency* based on repeated experiments. We would also like to acknowledge that *Uncertainty Theory* and *Probability Theory* together form complementary mathematical systems, defining two different mathematical models to deal with uncertainty [16].

When collaborating with our industrial partners, frequencies of uncertainties are unavailable, but our industrial partners can provide subjective measurements of uncertainty (i.e., belief degree). This is the main reason why we employed *Uncertainty Theory*, instead of *Probability Theory,* to obtain uncertainty measurements. In addition, the way of handling uncertainty in *Uncertainty Theory* well fits our definition of uncertainty inherited from *U-Model* [13], by considering uncertainty from a human subjective perspective. In the literature, *Uncertainty Theory* has been applied to solve various problems, including optimal control [18], optimal scheduling (the train timetable problem [19]), reliability analysis [17], risk assessment [20], and the maximum flow problem of the network [21].

3 Running Example

To illustrate UncerTest, we present a running example about a simplified security system of the SafeHome system described in [22]. In general, the security system controls and configures alarms and some related sensors for implementing the various security and safety features, e.g., intrusion detection. Note that all text with underline can be referred to the running example (Fig. 1 -- Fig. 5).

Example 1. Belief Model. A key input of UncerTest are BM[1] specified by UncerTum (as discussed in Section 2.1), an example of which is shown in Fig. 1 (state machine) and Fig. 2 (class diagram). The class diagram represents the structure of the security system. For example, the security system is composed of an alarm, a set of sensors and a set of buttons. The state machine depicts a behavior of the security system, i.e., enabling the monitoring function for activating intrusion detections. Each state in the state machine requires an explicit state invariant, an example shown in the fragment *(A)* of Fig. 1, which can be used to derive a test oracle in the context of MBT. UncerTum distinguishes itself from other UML-based modeling solutions because it allows the construction of uncertainty. For instance, as shown in Fig. 1, uncertainty *u(S1, T2, S2)* represents that state *S2: Monitoring* might be reached from *S1: Idle* when a user presses a button, which triggers the occurrence of transition *T2: EnableMonitoring*. Corresponding to a situation that *S2: Monitoring* might not be reached, an uncertainty *u(S1, T2, S3)* is constructed in the model. In addition, an uncertainty measurement allows being specified in the model for representing how likely the uncertainty occurs, e.g., the measurement of *u(S1, T2, S2)}* being *0.98* as shown in the fragment *(B)* of Fig. 1.

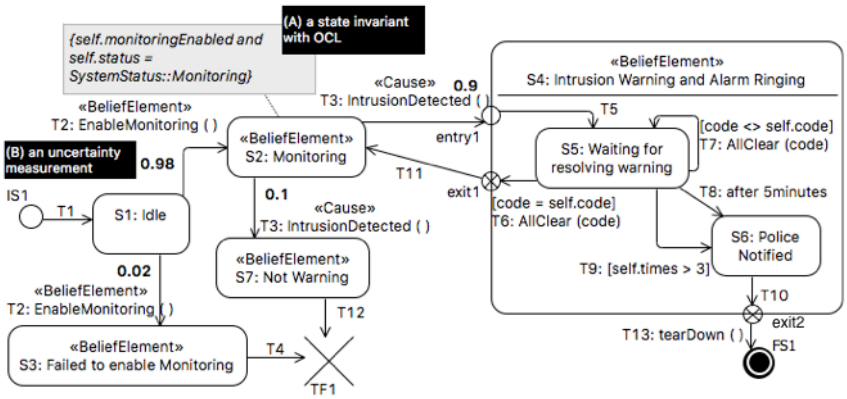


Fig. 1. An example of Belief State Machine

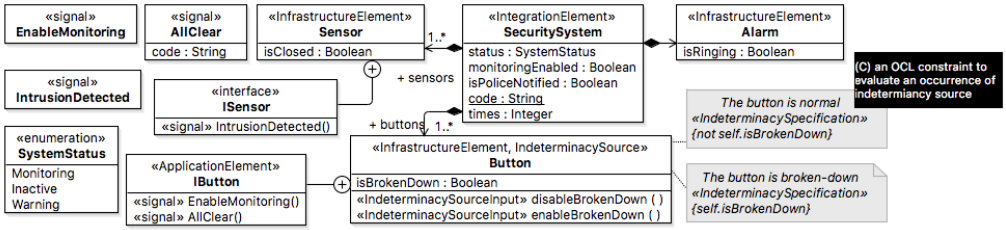


Fig. 2. An example of Belief Class Diagram

Example 2. Enabling IndeterminacySource. Regarding uncertainty *u(S1, T2, S2)*, one possible source (i.e., *IndeterminacySource*) of its occurrence may relate to the physical button. For instance, the status of the button is indeterminate (*isBrokenDown* in Fig. 2), either *broken-down* or *normal*. First, *(C)* of Fig. 2 explicitly presents an occurrence of the possible indeterminacy source using a constraint with OCL, i.e., *The button is normal*. In addition, *(D)* of Fig. 3 represents that an operation *disableBrokenDown()* (e.g., test API) can be used to ensure that the button is in the *normal* status. Moreover, *(E)* of Fig. 3 specifies a causal relationship between uncertainty and an indeterminacy source, i.e., *u(S1, T2, S2)* may occur when the button is *normal*. Last, *(F)* of Fig. 3 configures a strategy to enable related indeterminacy sources of uncertainty during test execution, i.e., *Always* enable the *Specified* related indeterminacy source (i.e., *normal*) on *Just Previous* of the uncertainty (i.e., *u(S1, T1, S2)*).

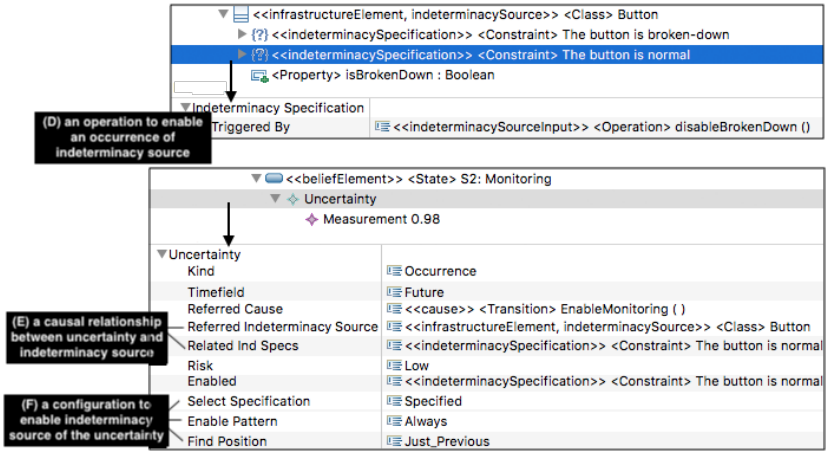
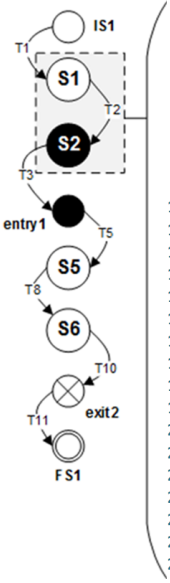


Fig. 3. An example of a configuration for enabling indeterminacy sources

Example 3. Abstract/Executable test case. With the models shown in Fig. 1, Fig. 2 and Fig. 3, an abstract test case can be derived (as shown in the left side of Fig. 4) from initial state *IS1* to final state *FS1* with two uncertainties: *u(S1, T2, S2)* and *u(S2, T3, S4)*. The corresponding executable code regarding the fragment *S1, T2* and *S2* is shown in the right side of Fig. 4. For instance, state *S1* is converted into *lines 1-3* (in Fig. 4), which evaluate the state invariant of *S1*, followed by an operation (*disableBrokenDown*) to enable specified indeterminacy source (*lines 4-6* in Fig. 4). The operation is inserted before an invocation of *T2*, which corresponds to configuration *Just Previous*. Also, the occurrence of the uncertainty and its alternative uncertainty together with its related indeterminacy source are also evaluated (*lines 10-23* in Fig. 4).




```
1  //evaluate state invariant: S1
2  assertTrue("S1: Idle", MOCLUtil.getInstance().evaluateOCL(system.eClass(),
3          "(not self.monitoringEnabled) and self.status = SystemStatus::Inactive", system));
4  //enable indeterminacy input: disableBroken
5  UncerTestReporter.reportTestStep("execute disableBrokenDown");
6  SafeHomeExecutionUtil.getInstance().opTestAPI("disableBrokenDown");
7  //execute transition:T2: EnableMonitoring
8  UncerTestReporter.reportTestStep("execute enableMonitoring");
9  SafeHomeExecutionUtil.getInstance().opTestAPI("enableMonitoring");
10 //uncertainty occurrence and its source evaluation
11 UncerTestReporter.setUncerInfo("<S1: Idle, T2: EnableMonitoring, S2: Monitoring>",
12         MOCLUtil.getInstance().evaluateOCL(system.eClass(),
13                 "self.monitoringEnabled and self.status = SystemStatus::Monitoring",
14                 system),
15         Arrays.asList("<The button is broken-down, not self.isBrokenDown, button>")
16                 .toArray(new String[1]),
17         MOCLUtil.getInstance().toboolenArray(Arrays.asList(MOCLUtil.getInstance().evaluateOCL(button.eClass(),
18                 "not self.isBrokenDown",  button))));
19 //alternative uncertainty occurrence evaluation
20 UncerTestReporter.setAlterUncerInfo("<S1: Idle, T2: EnableMonitoring, S2: Monitoring>",
21         "<S1: Idle, T2: EnableMonitoring, S3: Failed to enable Monitoring>",
22         MOCLUtil.getInstance().evaluateOCL(system.eClass(),
23         "self.monitoringEnabled and self.status = SystemStatus::Inactive", system), button, alarm);
24 //evaluate state invariant: S2
25 collector.checkThat("S3: Monitoring", MOCLUtil.getInstance().evaluateOCL(system.eClass(),
26         "self.monitoringEnabled and self.status = SystemStatus::Monitoring",system), IsEqual.equalTo(true));
```

*- Note that **entry1** (in the diagram) is an entry point of composite state S4. To flatten the composite state, **entry1** is used to represent S4.*

Fig. 4. An example representing the logic of an abstract and an executable test cases (partial) generated by UncerTest

Example 4. An assigned uncertainty verdict. An assigned uncertainty verdict of *u(S1, T2, S2)* is shown in Fig. 5. As seen from the results, the specified *u(S1, T2, S2)* occurred with its related indeterminacy source (*normal* button). In addition, none of its alternatives occurred. Thus, we identified that a known uncertainty occurred with its indeterminacy source, which is referred as *KnOccurred-With-IndS*. Note that an *unknown* uncertainty can be identified when none of the specified uncertainty and its alternatives occurred.

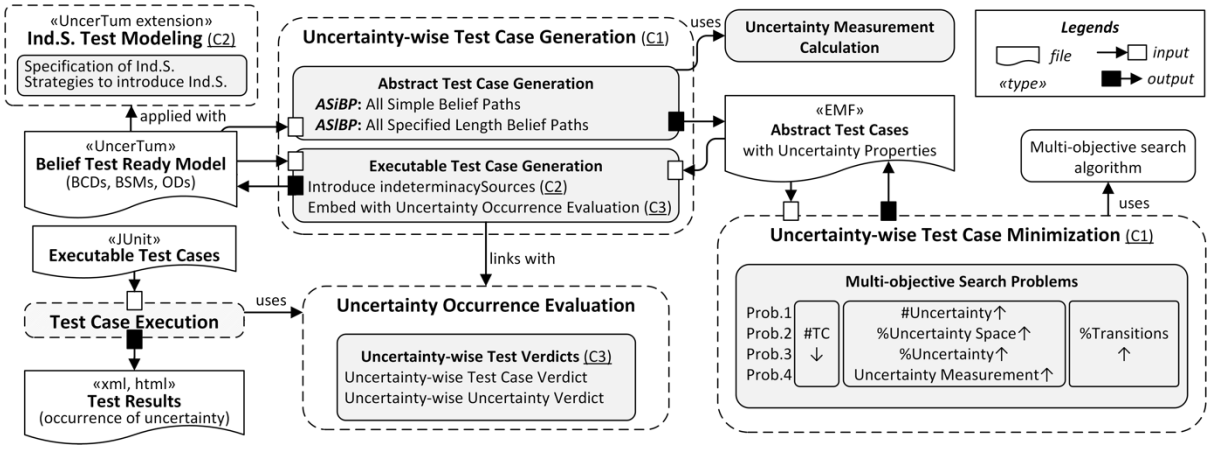

```
{
  "id": "<S1: Idle, T2: EnableMonitoring, S2: Monitoring>",
  "occurred": true,
  "alterUns": [
    {
      "id": "<S1: Idle, T2: EnableMonitoring, S3: Failed to enable Monitoring>",
      "occurred": false,
      "relatedIndSpecs": [
        {
          "id": "<The button is broken-down, self.isBrokenDown, button>",
          "occurred": false
        }
      ]
    }
  ],
  "relatedIndSpecs": [
    {
      "id": "<The button is normal, not self.isBrokenDown, button>",
      "occurred": true
    }
  ],
  "pars": [
    "system",
    "button",
    "alarm"
  ],
  "eobjs": [...],
  "curPar": "system",
  "uncertaintyResult": "KnOccurred-With-InS"
}
```


Fig. 5. An example of the occurrence result of the uncertainty, *u(S1, T2, S2)*

4 Overview of UncerTest

Fig. 6 shows an overview of UncerTest that mainly consists of four components: uncertainty-wise test generation,

uncertainty-wise test minimization, indeterminacy source test modeling, and uncertainty-wise test verdicts.



Note that *C1, C2* and *C3* are the scientific challenges in Table 1.

-*Ind.S.*: IndeterminacySource, *BCDs*: Belief Class Diagrams, *BSMs*: Belief State Machines, and *ODs*: Object Diagrams.

Fig. 6: Overview of UncerTest

To address the scientific challenge of C1(Table 1), we employ models that capture expected behaviors and known uncertainties of a CPS under test as a reference to generate test cases systematically and automatically (i.e., **uncertainty-wise test generation**). As shown in Fig. 6, BMs[1] are such models developed using UncerTum (Section 2.1) for enabling uncertainty test modeling. UncerTest has two test case generation strategies corresponding to two coverage criteria: *ASiBP* and *ASlBP*. These two strategies are inspired from [23], are designed particularly for belief state machines, and considered the uncertainty aspects such as the number of uncertainties in a test path and overall uncertainty measurement of a test path. Moreover, UncerTest considers advanced features of UML state machines such as composite states, submachine states, and orthogonal regions. To cost-effectively minimize test cases for execution, UncerTest relies on multi-objective search to minimize generated test cases (i.e., **uncertainty-wise test minimization**). First, we apply two typical cost-effective objectives: *PerTMin* (*percentage-of-test-case-minimization*) to reduce the number of test cases, and *PerTransition* (*percentage-of-transition-coverage*) to maintain coverage of BMs. Additionally, we newly define four uncertainty related objectives: *AvgNU* (*average-normalized-number-of-uncertainties-covered*) about the quantity of uncertainties (to maximize the number of uncertainties), *PerUSpace* (*percentage-of-uncertainty-space-covered*) regarding the uncertainty space (to maximize uncertainties that are from different uncertainty spaces), *AvgUM* (*average-overall-uncertainty-measurement*) regarding uncertainty measurement (to maximize uncertainties that highly occur), and *PerUniqueU* (*percentage-of-unique-uncertainties-covered*) regarding diversity (to maximize uncertainties that are different). Based on these six objectives, we formulated four multi-objective search problems (Prob.1 – Prob. 4 in Fig. 6), each of which was defined with *PerTMin*, *PerTransition* and one of the four uncertainty-related objectives.

To increase chances of occurrences of (known or *unknown*) uncertainties (to address the scientific challenge of C2 in Table 1), one viable way is to enable occurrences of uncertainty sources (i.e., *IndeterminacySources*). In UncerTest, such sources are captured in BMs with the **indeterminacy source test modeling** methodology (Fig. 6), an extension of UncerTum. This extension enables a detailed construction of an indeterminacy source including specifying constraints to evaluate its occurrences and a property referring a test API to enable its occurrence. Since one uncertainty might have multiple sources, we need a strategy to decide which indeterminacy source to introduce. Besides, an indeterminacy source may lead to more than one uncertainty, thus, we also need to decide where to introduce the indeterminacy source such that the occurrence of the concerned uncertainty (not the others sharing the same indeterminacy source) can be enabled. To achieve these objectives, we propose a set of strategies to enable the decision making of which sources to introduce, where to introduce them and how to introduce them during test generation. Subsequently, these sources are carried as parts of executable test cases. Based on different configurations of the strategies of introducing indeterminacy sources , a CPS can therefore be tested under different combinations of them. For instance, Fig. 4 shows an executable test case generated with indeterminacy sources according to the configuration shown in Fig. 3.

To address the scientific challenge of C3 (Table 1), we are concerned with discovering *unknown occurrences* of uncertainties (i.e., uncertainties previously unknown or known uncertainty occurred with unknown sources) during testing. Thus, a test result of a test case should explicitly present occurrences of uncertainties and their sources covered by the test case. To make such occurrences observable, we propose **uncertainty-wise test verdicts** (Fig. 6), which includes an integrated evaluation of 1) the occurrence of a specified uncertainty covered in the test case, 2) the occurrence of related sources, and 3) the occurrence of alternatives of the uncertainty. With

such verdicts, an *unknown* uncertainty can be identified, i.e, when none of the specified (known) uncertainties and their alternatives occurred during test execution; and, an *unknown* indeterminacy source can also be identified, i.e., when none of the indeterminacy sources occurs, but the uncertainty occurs. Moreover, based on test results with the verdicts, further analyses can also be supported. For instance, a causal relationship between uncertainty and its indeterminacy sources can further be quantified as, e.g., one source leads to the occurrence of an uncertainty with the 80% probability. Information like this may help to provide additional guidance on how to operate the system by for example reducing a chance of occurrences of an indeterminacy source.

5 Abstract Test Case Generation

UncerTest automatically generates abstract test cases from belief models with the test case generation strategies. In Section 5.1, we provide definitions of key concepts. A comprehensive list of definitions of concepts is however provided in Appendix A for reference. In Section 5.2, we discuss the uncertainty measurement calculation, followed by the generation strategies (Section 5.3).

5.1 Definitions

In a Belief State Machine, uncertainty is a situation whereby the belief agent does not have full confidence that a state s_x transits to another state s_y through a transition t_z, represented as $u(s_x, t_z, s_y)$. In addition, an uncertainty space is a set of uncertainties that originate from the same state s_x with the same transition t_z , represented as $usp(s_x, t_z)$. As shown in Fig. 7, *u(S1, T2, S2)* is an uncertainty, and *u(S1, T2, S2)* and *u(S1, T2, S3)* belong to the same uncertainty space: *usp(S1, T2)*.

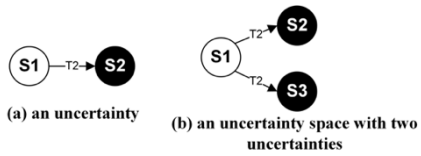
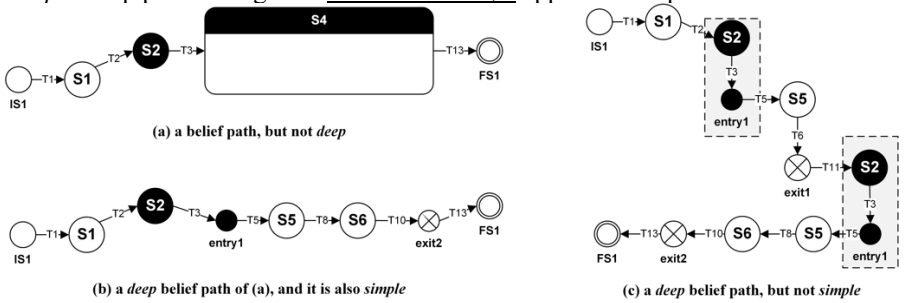


Fig. 7: Illustrating uncertainties and uncertainty spaces from the Belief State Machine shown Fig. 1

An abstract test case based on a state machine can be represented as a *path* of traversing the state machine from an initial state to a final state. In UncerTest, such an abstract test case is derived from a belief state machine, and the corresponding path is called a *belief path*, as it contains uncertainty information. As a belief state machine is essentially a UML state machine, a composite state may exist in a *belief path*; therefore, it is needed to further flatten such a composite state. Subsequently, in UncerTest, we propose *deep belief paths*, i.e., *belief paths* that do not have any composite or submachine state, and each *deep belief path* is an abstract test case. For example, Fig. 8 (a) is a *belief path* but not *deep*, and Fig. 8 (b) demonstrates a *deep belief path* derived from the belief state machine in Fig. 7. Furthermore, we define *simple deep path*s as *belief paths* that only contain unique states and transitions, e.g., the *deep belief path* shown in Fig. 8 (b) is also *simple*. Conversely, a path shown in Fig. 8 (c) is therefore not a *simple* deep path as fragment $\underline{S2 \rightarrow T3 \rightarrow entry1}$ appear in the path twice.



IS*: Initial State,* S*: State,* FS*: Final State, and* T*: Transition,* black filled circle: *State stereotyped with «BeliefElement»*

Fig. 8: Illustrating *belief paths*

Since an abstract test case (i.e., a *deep belief path*) is derived from a belief state machine, we further extract a set of its properties: 1) a multi-set of uncertainties covered, 2) a set of unique uncertainties covered, 3) a set of unique uncertainty space covered, 4) a set of unique transitions covered, and 5) an uncertainty measurement of abstract test case.

5.2 Uncertainty Measurement Calculation

To calculate the uncertainty measurement of an abstract test case, we required that each uncertainty in a Belief State Machine should be specified with uncertainty measurement, represented as *um.* In this section, we discuss basic concepts of probability theory and uncertainty theory. In addition, an example to demonstrate how to apply these theories in UncerTest are presented in Table 2 and Fig. 9.

(1) Probability theory

To apply Probability theory, measurements of uncertainties (i.e., frequency) should follow three axioms and product probability theorem of probability theory:

Axiom 1. (Normality) $Pr(\Omega) = 1$, (Ω is the universal set).

Axiom 2. (Nonnegativity Axiom) Pr $(A) \geq 0$, where A is any event in Ω.

Axiom 3. (Additivity Axiom) $Pr\{\bigcup_{i=1}^{\infty} \mathrm{A}_i\} = \sum_{i=1}^{\infty} Pr\{\mathrm{A}_i\}$ (every *countable* sequence of mutually disjoint events $\mathrm{A}_1, \mathrm{A}_2, ...$).

Theorem (Product Probability): Let (Ω_k, A_k, Pr_k) be probability spaces for $k = 1, 2, \ldots$. The product probability is a probability measure satisfying $Pr\{\bigcap_1^{\infty} A_k\} = \prod_1^{\infty} Pr\{A_k\}$, where A_k is arbitrarily chosen events from A_k for $k = 1, 2, \ldots$ respectively.

(2) Uncertainty theory

Uncertainty Theory defines a term called *Uncertainty Measure (*represented as the $\mathcal{M}$ symbol*)*, which captures a specific uncertainty value (a number) related to an event. This number assigns the *belief degree* [13] of a *belief agent* [13] to the event, to indicate her/his confidence about the occurrence of the event [17]. As Liu suggested in [17], $\mathcal{M}$ respects the following four axioms:

Axiom 1. (Normality) $\mathcal{M}(\Gamma) = 1$, (Γ is the universal set).

Axiom 2. (Duality) $\mathcal{M}\{\Lambda\} + \mathcal{M}\{\Lambda^c\} = 1$, where Λ shows a particular event, whereas Λ^c shows all the elements in the universal set excluding Λ.

Axiom 3. (Subadditivity) $\mathcal{M}\{\bigcup_{i=1}^{\infty} \Lambda_i\} \leq \sum_{i=1}^{\infty} \mathcal{M}\{\Lambda_i\}$ (every *countable* sequence of events $\Lambda_1, \Lambda_2, ...$).

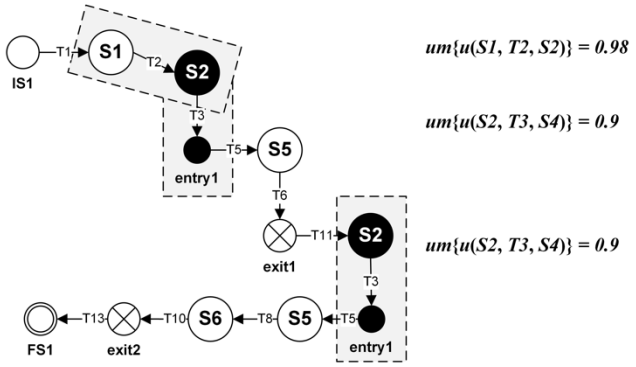
Uncertainty Space: A triplet $(\Gamma, \mathcal{L}, \mathcal{M})$, where Γ is the universal set, $\mathcal{L}$ is a σ-algebra [24] over Γ, and $\mathcal{M}$ is *Uncertainty Measure.*

Theorem: Let $(\Gamma_k, \mathcal{L}_k, \mathcal{M}_k)$ be uncertainty spaces and $\Lambda_k \in \mathcal{L}_k$, for $k = 1, 2, \ldots n$. Then $\Lambda_1, \Lambda_2, \ldots \Lambda_n$ are always independent of each other if they are from different uncertainty spaces.

Axiom 4. (Product Axiom) Let $(\Gamma_k, \mathcal{L}_k, \mathcal{M}_k)$ be uncertainty spaces for $k = 1, 2, \ldots$. The product uncertainty measure is an uncertainty measure $\mathcal{M}$ satisfying $\mathcal{M}\{\prod_{i=1}^{\infty} \Lambda_i\} = \bigwedge_{k=1}^{\infty} \mathcal{M}\{\Lambda_k\}$, where Λ_k is arbitrarily chosen events from $\mathcal{L}_k$ for k = 1, 2, … respectively.

Table 2. An example to calculate uncertainty measurement using probability theory and uncertainty theory

Concepts	Probability Theory	Uncertainty Theory
usp(S, T)	Universal set, Ω **Example.** Ω = {*u(S1, T2, S2), u(S1, T2, S3)*}	Universal set, Γ **Example.** Γ = {*u (S1, T2, S2), u(S1, T2, S3)*}
um of an uncertainty	*um(u) = ProbabilityMeasure(u) = Pr(u) = frequency = times to occur / times to sample* **Example.** *Pr{u(S1, T2, S2)}=0.98*, *Pr{u(S1, T2, S3)}*= 0.02	*um(u) = UncertaintyMeasure(u) = $\mathcal{M}$ {u}= belief degree of U specified by domain experts* **Example.** $\mathcal{L}$ = {∅, {*u (S1, T2, S2)*} {*u(S1, T2, S3)*}, Γ}, $\mathcal{M}$ {∅} = 0, $\mathcal{M}$ {*u(S1, T2, S2)*} =0.98, $\mathcal{M}${*u(S1, T2, S3)*} = 0.02 and $\mathcal{M}${Γ}=1
um of a test case	$Pr\{\bigcap_1^{\infty} A_i\} = \prod_1^{\infty} Pr\{A_i\}$ **Example.** *Pr(t) = Pr{u(S1, T2, S2)}×Pr{u(S2, T3, S4)}×Pr{u(S2, T3, S4)}* = *0.98×0.9×0.9= 0.048*	$\mathcal{M}\{\prod_{i=1}^{\infty} \wedge_i\} = \bigwedge_{k=1}^{\infty} \mathcal{M}\{\wedge_k\}$ **Example**. *$\mathcal{M}$(t) = $\mathcal{M}$ {u(S1, T2, S2)}∧ $\mathcal{M}$ {u(S2, T3, S4)}* ∧ *$\mathcal{M}$ {u(S2, T3, S4)}* = *0.98∧ 0.9∧0.9= 0.9*



Note that *entry 1* is the entry point of composite state *S4*, and entry 1 represents *S4* in a flatten path, i.e., deep belief path

Fig. 9: Illustrating abstract test cases with uncertainties and their *UM*s

5.3 Test Case Generation Strategies

In the literature, some test case generation strategies based on state machines have been proposed including *All Transitions*, *All States,* and *All Predicates* [25-28]. For UncerTest, we propose two test case generation strategies, inspired by *Prime Path Coverage* and *Specified Path Coverage*, both presented in [23].

All Simple Belief Path Coverage (ASiBP): Test set *T* satisfies *ASiBP* on a belief state machine if and only if any belief simple deep path from the *initial state* to one of the *final state*s in the belief state machine is in *T*. As said in [23], "One useful aspect of the simple path is that any path can be created by composing simple paths".

We propose *ASiBP* to cover all minimal paths, based on which any path-based coverage criterion can be defined by extending a path generated with *ASiBP* (i.e., side trips and detours [23]). The test set generated using this strategy is the cross product of all the possible *simple deep* belief paths across all the regions.

All Specified Length Belief Path Coverage (ASlBP): Test set *T* satisfies *ASlBP* on a belief state machine if and only if the length of any belief simple deep path less than the *specified length* from the *initial state* to one of the *final state*s in the belief state machine is in *T*. We propose *ASlBP* because it can be configured 1) for specific needs (e.g., saving cost by generating less numbers of test cases), 2) to subsume *All Transitions*, *All States,* and *All Predicates* when needed, 3) to generate a larger size of test set from a belief model (which are more diverse in terms of attached uncertainty information) to form a better pool for test minimization, and 4) to subsume the *All Uncertainty* coverage, which we define as covering all states and transitions with uncertainty. The test set generated with this strategy consists of all possible *deep* belief paths with loops allowed, and all the lengths of these paths should not be longer than the maximum allowed length, which is configurable (as discussed above) and should be pre-defined before applying the strategy. For example, one way of defining the maximum allowed length for generating paths for a region is to calculate the total number of states and transitions contained in it.

6 Uncertainty-Wise Test Case Minimization

6.1 Problem Representation

Depending on which test case generation strategy to apply, how it is configured (for *ASlBP*) and how complex a CPS under test is, the number of generated abstract test cases can potentially be very large and it would be practically impossible to execute executable test cases generated from all of the abstract test cases within a limited time budget. It is, therefore, important to minimize the number of abstract test cases based on various attributes associated with each test case.

Let $T=\{t_i \mid 0<i<nt\}$ be a test set derived from a belief state machine with one of the UncerTest test generation strategies. Each test case *t* has four attributes related uncertainty (Section 5.1). $S = \{s_1, \ldots, s_{ms}\}$ presents a set of potential solutions, i.e., a subset of *T*, where *ms* is the total number of potential solutions and is calculated as 2^{nt}-1 except that the solution selects none of the test cases. As the number of generated test cases increases, the search space will increase exponentially. For any test case minimization problem, solution *s* contains a set of selected test cases, formalized as $T_{sub}=\{t_j' \mid 0<j<mt,\ t_j' \in T\} \subseteq T$, where *mt* is the number of selected test cases. Each solution *s* is characterized by a set of values of cost and effectiveness measures. In UncerTest, we defined six objectives and four uncertainty-wise multi-objective minimization problems with consideration of three aspects: cost, effectiveness, and uncertainty. Each of these four problems is composed of three of the six objectives.

6.2 Definitions of the Minimization Objectives

We define six minimization objectives: cost measure O1, four uncertainty related measures (O2-O4), and effectiveness measure O6.

O1. Percentage of Test Case Minimization (*PerTMin*): *PerTMin* is the percentage of selected test cases in solution T_{sub}:$PerTMin = \frac{mt}{nt} \times 100\%$, where nt and mt are the numbers of test cases in *T* and T_{sub}.

O2. Average Normalized Number of Uncertainties Covered (*AvgNU*): *AvgNU* measures the average normalized number of uncertainties covered by the selected test cases of a solution. For each test case t_i', the number of uncertainties covered is $nu(t_i')$, which can be normalized [1] as: $nor(nu(t_i')) = \frac{nu(t_i')}{nu(t_i')+1}$. *AvgNU* for the selected test cases is $\frac{\sum_{i=1}^{mt} nor(NU(t_i'))}{mt}$, where *mt* is the number of test cases in T_{sub}.

O3. Percentage of Uncertainty Space Covered (*PerUSpace*): *PerUSpace* measures the percentage of the total set of uncertainty spaces of a Belief State Machine covered by the selected test cases of a solution. Suppose, the set of uncertainty spaces of the state machine is $USP_{SM} = \{usp_i \mid 0 < i \leq nusp\}$ and the set of uncertainty spaces of the selected test cases is the intersection of the uncertainty spaces across each test case t_i', $USP_{sub} = \bigcap_i^{mt} USP_{t_i'} = \{usp_i' \mid 0 < i \leq musp\} \subseteq USP_{SM}$. *PerUSpace* is then defined as: $\frac{musp}{nusp} \times 100\%$.

O4. Average Overall Uncertainty Measurement (*AvgUM*): *AvgUM* is the overall uncertainty measurement of the selected test cases of a solution. Note that for test case t_i', $um(t_i')$ is an uncertainty measurement calculated based on any of the two applicable mathematical theories (Section 5.2). Thus, $AvgUM = \frac{\sum_{i=1}^{mt} um(t_i')}{mt} \times 100\%$.

O5. Percentage of Unique Uncertainties Covered (*PerUniqueU*): *PerUniqueU* measures the percentage of the total number of unique uncertainties covered by the selected test cases of a solution. Suppose that the set of unique uncertainties in a Belief State Machine is $UU_{SM} = \{u_i \mid 0 < i \leq nuu\}$ and the set of unique uncertainties of the selected test cases is the interaction of the unique uncertainties across each test case t_i', $UU_{sub} = \bigcap_i^{mt} UU_{t_i'} = \{u_i' \mid 0 < i \leq muu\} \subseteq UU_{SM}$, then *PerUniqueU* is calculated as: $\frac{nuu}{nuu} \times 100\%$.

O6. Percentage of Transition Coverage (*PerTransition*): *PerTransition* measures the percentage of the total

number of transitions in a Belief State Machine covered by the selected test cases of a solution. Suppose that *ntr* is the total number of transitions in a Belief State Machine, and *mtr* is the number of transitions in the selected test cases (the size of the interactions among the transition sets of each selected test case t_i', $Transition_{sub} \cap_i^{mt} Transition_{t_i'} = \{tr_i' | 0 < i \leq mtr\}$). *PerTransition* is calculated as: $\frac{mtr}{ntr} \times 100\%$.

6.3 Uncertainty-wise Test Case Minimization Problems

To reduce the number of test cases to execute and maximize the coverage of transitions in test ready models, *PerTMin* and *PerTransition* are two necessary objectives. Further, we define the following four test case minimization problems that minimize *PerTMin*, maximize *PerTransition*, and at the same time achieve four distinct uncertainty-related objectives.

Problem 1. Search for a solution T_{sub} to achieve: 1) low *PerTMin* ; 2) high *AvgNU*; and 3) high *PerTransition.* We define *Problem 1* to select the minimum number of test cases to cover the maximum number of known uncertainties possible, aiming to observe the reaction of the CPS in the presence of a maximum number of uncertainties with the minimum possible test cases.

Problem 2. Search for a solution T_{sub} to achieve: 1) low *PerTMin*; 2) high *PerUSpace*; and 3) high *PerTransition.* We define *Problem 2* to select the minimum number of test cases to cover at least one uncertainty from each uncertainty spaces. We aim to observe the reaction of the CPS in the presence of uncertainties from all known uncertainty spaces with the minimum possible test cases.

Problem 3. Search for a solution T_{sub} to achieve: 1) low *PerTMin*; 2) high *AvgUM*; and 3) high *PerTransition*. We define *Problem 3* to select the minimum number of test cases to maximize the coverage of the parts of the system with a high degree of confidence.

Objective *AvgUM* prefers higher values because: 1) an occurrence of uncertainty is prerequisite to test CPSs with uncertainty. If an uncertainty measurement is quite low, there is a higher chance of testing CPSs without uncertainties; and 2) a higher value normally reflects an expected system behavior. For example, an expected behavior of the running example (Fig. 1) is successfully enabling monitoring (i.e., *u(S1, T2, S2)*), and its measurement is 98%.

Problem 4. Search for a solution T_{sub} to achieve: 1) low *PerTMin*; 2) high *PerUniqueU*; and 3) high *PerTransition*. We define *Problem 4* to select the minimum number of test cases and to maximize the coverage of different uncertainties. We aim to test the behavior of a CPS under diverse uncertainties with the minimum number of test cases.

7 Executable Test Case Generation

In UncerTest, generating executable test cases from abstract test cases (Section 5) is mainly about how to generate test data and introduce indeterminacy sources (i.e., sources of uncertainties), specified in test ready models.

7.1 Enabling Indeterminacy

Since we focus on testing a CPS in the presence of environmental uncertainties, we need to introduce uncertainties in the physical environment, which may lead to uncertain behaviors of the CPS. To achieve this, we need to model such environmental uncertainties (named as "Indeterminacy Sources" for being more precise).

Fig. 10 shows part of the UUP profile (Section 2.1) for modeling indeterminacy sources. We provide a set of options to model indeterminacy sources, e.g., as a UML Operation and a constraint specified in Object Constraint Language (OCL) [29]. An indeterminacy source always has 1..* indeterminacy specifications, i.e., «IndeterminacySpecification» (conditions) that must be true for an indeterminacy source to occur. «IndeterminacySourceInput» specifies the action that triggers the occurrence of «IndeterminacySource».

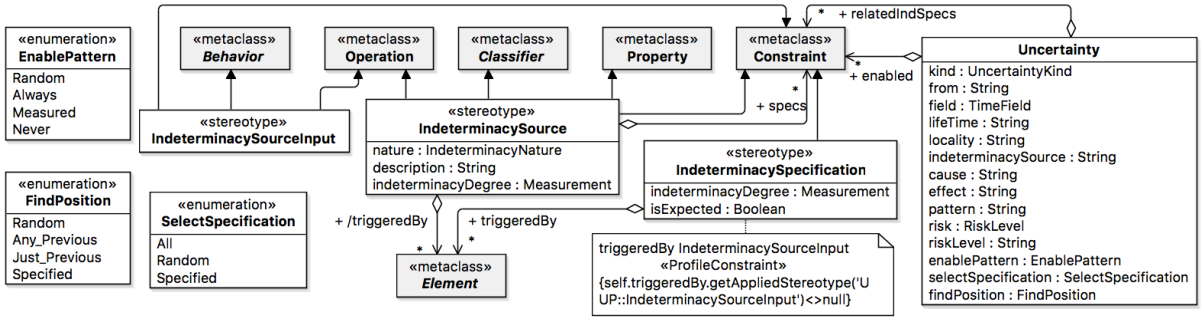


Fig. 10: UML Profile Diagram of IndeterminacySource (Partial)

It is possible to model these indeterminacy-related concepts in different ways. Therefore, to ease the modeling process, we summarize our recommendations for applying this part of the profile in Fig. 10, based on our

experience. For example, in the first situation (as described as *Case1* in Table 3), we recommend modeling an indeterminacy source as a UML *Property*, when states of a CPS or its environment can be directly accessed and are indeterminate.

Note that for the first and third situations (*Case1* and *Case3* in Table 3), we recommend specifying an indeterminacy source input either as an Operation without parameters (*Option1*) or as an Operation with parameter(s) constrained with an OCL constraint (*Option2*). Also, for *Case1* and *Case3*, an indeterminacy source can be specified as a property (*Rule1.1 and Rule3.1*) or constraint (*Rule1.2 and Rule 3.2*). If it is *Rule1.2 or Rule3.2*, its corresponding indeterminacy specification(s) can then be simply specified as *FALSE* by default and must be switched to *TRUE* to enable the related indeterminacy source.

Table 3: Recommendations for applying an indeterminacy source

#	Stereotype Applied		Base Element
Case1: States of the environment of the CPS are indeterminate (e.g., the status of the button).			
Rule1.1	«IndeterminacySource»		Property
	«IndeterminacySpecification»		Constraint
	Option1	«IndeterminacySourceInput»	Operation
	Option2	«IndeterminacySourceInput»	Operation, Constraint
Rule1.2	«IndeterminacySource»		Constraint
	«IndeterminacySpecification»		FALSE (default)
	Option1	«IndeterminacySourceInput»	Operation
	Option2	«IndeterminacySourceInput»	Operation, Constraint
Case2: Input data is indeterminate.			
Rule2.1	«IndeterminacySource»		Operation
	«IndeterminacySpecification»		Constraint
	«IndeterminacySourceInput»		Constraint
Case3: Occurrences of an event from the environment (e.g., "pressing the button") are indeterminate.			
R3.1	«IndeterminacySource»		Property
	«IndeterminacySpecification»		Constraint
	Option1	«IndeterminacySourceInput»	Operation
	Option2	«IndeterminacySourceInput»	Operation, Constraint
R3.2	«IndeterminacySource»		Constraint
	«IndeterminacySpecification»		FALSE (default)
	Option1	«IndeterminacySourceInput»	Operation
	Option2	«IndeterminacySourceInput»	Operation, Constraint

SelectSpecification and *FindPosition* (Fig. 10) enable an indeterminacy source associated with a specific uncertainty, their corresponding indeterminacy specifications, and inputs during test execution. *EnablePattern* (Fig. 10) provides four ways of enabling an indeterminacy source: 1) *Random* – the indeterminacy source is introduced randomly (from the uniform random distribution) during execution; 2) *Always* - the indeterminacy source is always enabled during execution; 3) *Measured* - the indeterminacy source is enabled during execution by a specified measurement, e.g., with a normal distribution; and 4) *Never* - the indeterminacy source is never enabled during the execution. Choosing which option depends on how much knowledge and information (e.g., experience, historical data) one has about the system.

SelectSpecification provides three ways of selecting which indeterminacy specification(s) of an indeterminacy source to be enabled during test execution: 1) *All* – all associated indeterminacy specifications are enabled; 2) *Random* – enable a random number of randomly selected specification(s) from all the specifications associated with the indeterminacy source during test execution; and 3) *Specified* – the indeterminacy specification(s) specified with the "enabled" attribute is enabled during the test execution. Similarly, which option to take is highly dependent on users' experience, knowledge and available historical data.

FindPosition is about finding a position of a path generated by UncerTest, in which an indeterminacy source should be enabled. We define four options for *FindPosition*: 1) *Random* - the position is generated randomly; 2) *Any_Previous* – the position can be any previous position before the occurrence of the associated uncertainty; 3) *Just_Previous* – the position is exactly the position right before the occurrence of the associated uncertainty; and 4) *Specified* – the exact position is modeled in the test ready model. *Random* is recommended when we have no particular preferences or guidance. *Any_Previous* is recommended when one wants to test, if possible, whether the uncertainty is actually due to the indeterminacy source enabled. *Just_Previous* should be used when one wants to know whether the occurrence of the uncertainty is due to its previous step. *Specified* should be used when one has a specific position in mind, based on for example previous experience or historical data.

Note that the three mechanisms can be configured by users to form a concrete strategy (as part of an overall test strategy) for enabling an indeterminacy source associated with an uncertainty and all or part of its associated indeterminacy specifications, at a particular position of a path, which is eventually transformed into executable test cases and executed. ***Example 2*** (Fig. 3) represents a configuration of enabling indeterminacy sources, and ***Example 3*** (Fig. 4) is a partial executable test case according to the configuration (Fig. 3).

7.2 Test Setup and Test Data Generation

When generating executable test cases, test configuration and concrete test data are needed. When applying

UncerTum, test configuration is recommended to be specified as a UML object diagram organized in a package. All the objects and their relationships in this test configuration package will be instantiated before execution.

First, test data generation is needed for triggering call events on transitions. In this case, a guard condition (an OCL constraint) on a transition specifies the valid set of values, with which the call event can be invoked. We used an existing test data generation tool called EsOCL [30], which takes an OCL constraint as an input and generates a set of values that satisfy the constraint. These values are then used as test data in executable test cases. Second, test data might be needed to trigger occurrences of indeterminacy sources. For any indeterminacy source input that is specified as a stereotyped Constraint or as a stereotyped Operation with its parameters constrained with a constraint, we rely on the EsOCL tool [30] to solve the constraint to generate test data. For any indeterminacy source input specified as an operation with no any parameter, no data needs to be generated to trigger the operation and hence the indeterminacy input.

8 Uncertainty-wise Test Verdicts

In state-based testing, a test result is typical determined by evaluating whether a specified state invariant is satisfied (i.e., *fail* or *pass*) with actual data specifying the state of a CPS. In the context of uncertainty-wise testing, a test result requires to carry additional information related to uncertainty such as an occurrence of uncertainty and an occurrence of introduced indeterminacy sources. In UncerTest, an uncertainty (s_x, t_z, s_y) is counted as *occurred* during testing, if and only if a state of the CPS under test changes from s_x to s_y by sending a stimulus according to transition t_z, in the sequence of checking that state invariant of s_x is satisfied, transition t_z is executed, and state invariant of s_y is satisfied. Regarding an occurrence of an indeterminacy source, it can similarly be determined by evaluating if its specification (i.e., an OCL constraint) is satisfied with run-time data from the CPS under test or test infrastructure. ***Example 3*** in Fig. 4 illustrates such evaluations.

Identifying unknowns is concerned with capturing an unknown occurrence of an uncertainty (i.e., the uncertainty is previously unknown, or the known uncertainty occurred with previously unknown sources) during testing. To identify such unknown occurrences, we design **uncertainty-wise test verdict**, with a comprehensive evaluation of occurrences of uncertainties and their sources, as shown in Fig. 11 (the conceptual model) and Table 4 (definitions). There are two types of uncertainty-wise test verdicts. The first type, i.e., *UncerVerdict*, defines verdicts for a set of possible evaluations of the occurrence of an uncertainty (i.e., test oracle). An *UncerVerdict* is determined by the result of an evaluation of an occurrence of an uncertainty, related sources of the uncertainty, and the alternatives of the uncertainty. The seven kinds of *UncerVerdict* are listed as the seven literals of enumeration *UncerVerdictKind* (definitions in Table 4), which correspond to an identification of unknown indeterminacy sources and uncertainties. An example (***Example 4***) of a result (i.e., *KnOccurred-With-InS*) of an uncertainty is shown in Fig. 5, and *KnOccurred-With-InS* is assigned to the uncertainty because the occurrence of the uncertainty is evaluated as TRUE, the occurrence of the related source is evaluated as TRUE, and meanwhile the occurrences of all the alternatives of the uncertainty are FALSE. The pseudocode to determine the verdict of an uncertainty with *UncerVerdict* is provided in Fig 12.

The second type of uncertainty-wise test verdict, i.e., *UncerTestCaseVerdict*, is defined in terms of a test case that contains uncertainties, by extending classical test case verdicts (e.g., *pass*). As shown in Fig. 11, an *UncerTestCaseVerdict* is specified as a sequence of *UncerVerdict*s specifying a set of possible evaluations of a test case, including the uncertainty aspect (e.g., known uncertainty occurred (i.e., *KnOccurred*)) and classical test case verdicts (e.g., *Pass*). The uncertainty related verdicts of a test case can be derived based on the verdicts of uncertainties in the test case. For instance, a test case is evaluated to be *KnOccurred* when at least one known uncertainty (with any of the three *KnOccurred-** types of *UncerVerdictKind*) occurred, and none of the verdicts of the uncertainties is *UkOccurred*. Definitions of *UncerVerdictKind* are shown in Table 4.

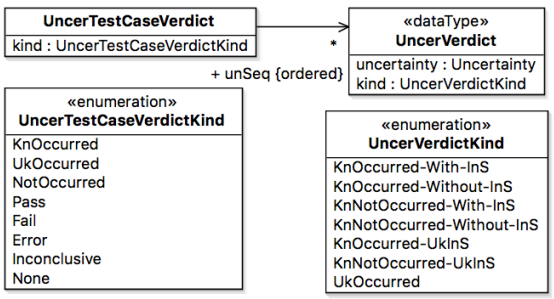


Fig. 11: Uncertainty-wise Test Verdicts – Conceptual Model

Table 4: Uncertainty-wise Test Verdicts – Definitions of the Literals of the Enumerations (Fig. 11).

Literal	Definition	Unknown detection
UncerVerdictKind: Kinds of verdicts for an uncertainty		
KnOccurred-With-InS	Known uncertainty occurred under the occurrence of a specified indeterminacy source	-
KnOccurred-Without-InS	Known uncertainty occurred under the non-occurrence of any specified indeterminacy source	**Unknown indeterminacy source**
KnNotOccurred-With-InS	Known uncertainty did not occur under the occurrence of any specified indeterminacy source. Meanwhile, at least one of alternatives of the uncertainty occurred.	
KnNotOccurred-Without-InS	Known uncertainty did not occur under the non-occurrence of any specified indeterminacy source. Meanwhile, at least one of alternatives of the uncertainty occurred.	-
KnOccurred-UkInS	Known uncertainty occurred, but its related indeterminacy source is unknown.	**Unknown indeterminacy source**
KnNotOccurred-UkInS	Known uncertainty did not occur, and its related indeterminacy source is unknown.	
UkOccurred	Known uncertainty did not occur, and none of its alternatives occurred.	**Unknown uncertainty**
UncerTestCaseVerdictKind: Kinds of the verdicts for a test case		
KnOccurred	At least one known uncertainty (with any of the three *KnOccurred-** types of *UncerVerdictKind*) occurred but no *UkOccurred*.	
UkOccurred	At least one *UkOccurred*.	
NotOccurred	All uncertainties are evaluated to be any of the three *KnNotOccurred-** kinds of *UncerVerdictKind*.	
Pass	The test case execution result, for which no uncertainty is specified, adheres to the expectations [31].	
Fail	The test case execution result, for which no uncertainty is specified, differs from the expectations [31].	
Error	An error is detected.	
Inconclusive	The test case execution result cannot be classified as *Pass*, *Fail*, *Error*, *KnOccurred*, *UkOccurred* or *NotOccurred*.	
None	A test case has not yet been executed.	

```
To Assign UncerVerdict for an uncertainty (agUV)
Input U = (sts, tr, stt)
 1  if eva(stt)
 2      if len(getrelatedIndSp(U)) = 0
 3          return KnOccurred-UkInS
 4      else
 5          for Indp: getrelatedIndSp
 6              if eva(Indp)
 7                  return KnOccurred-With-UkInS
 8          return KnOccurred-Wihtout-UkInS
 9  else
10      for altU: getAltUs(U)
11          if eva(altU.stt)
12              if len(getrelatedIndSp(U)) = 0
13                  return KnNotOccurred-UkInS
14              else
15                  for aIndp: getrelatedIndSp(altU)
16                      if eva(aIndp)
17                          return KnNotOccurred-With-InS
18                  return KnNotOccurred-Without-InS
19      return UkOccurred
```

Fig 12. Pseudocode to assign *UncerVerdictKind* for an uncertainty

9 Evaluation

Section 9.1 introduces case studies. Section 9.2 presents research questions. Section 9.3 presents the design of our evaluation. Results are presented in Section 9.4, the overall discussion is presented in Section 9.5, and threats to validity are presented in Section 9.6.

9.1 Case Study

To assess the cost-effectiveness of UncerTest, we selected two industrial CPS case studies. The first is GeoSports, and the system monitors the performance (e.g., speed and position) and health conditions of players both individually and as a team during a game with the ultimate objective of improving their performance. The GeoSports application that we tested is deployed for Bandy (a type of ice hockey commonly played in northern Europe) and uses the Quuppa system [32]. The testing infrastructure for Bandy is shown in Fig. 13. Instead of using real players to execute test cases, our industrial partner, Nordic Med Test [33] has deployed a set of test rigs for replacing players. Each test rig has one Quuppa device attached to it. The device communicates its position with one or more locators (antennas) via Bluetooth connections and the locators receive those positions and send them to the Quuppa Server (QPE). The access to the devices, locators, and the QPE server are available as REST APIs. Also, a set of test APIs was implemented by the partner as REST APIs for controlling the test rigs. Notice that we only tested the positioning system in this paper, i.e., collecting the positions from Quuppa tags and transmitting them to the QPE server via locators.

The second case study is Automated Warehouse (AW) provided by ULMA Handling Systems [12], Spain.

ULMA develops automated handling systems for worldwide warehouses of different natures such as Food and Beverages, Industrial, Textile, and Storage. Each handling facility (e.g., cranes, conveyors, sorting systems, picking systems, rolling tables, lifts, and intermediate storage) forms a physical unit, and together they are deployed to one handling system application (e.g., Storage). A handling system cloud supervision system (HSCS) interacts with diverse types of physical units, network equipment, and cloud services. Application-specific processes in HSCS are executed spanning clouds and CPS requiring different configurations. This case study implements several key industrial scenarios, i.e., introducing a large number of pallets to the warehouse, transferring the items by Stacker Crane. Instead of using real devices to test these scenarios, ULMA [12] and IK4-Ikerlan [34] developed and provided relevant simulators and emulators (Fig. 13). For example, two handling systems are deployed at two different sites (Site 1 and Site 2). For each site, the local superior monitors software and all types of devices and services and upload the data to the cloud superior through the network. Each physical device is developed as a simulator where the software, i.e., WMS and MFC, are deployed on. Also, a set of emulators are developed for manipulating the real physical environment, e.g., putting a pallet on the conveyor. To access the devices, software, and environment, the test APIs were implemented by the partner for controlling the physical device, sending requests to the software, and manipulating the physical environment. Further details on the case studies can be consulted in [35].

The descriptive statistics of the test ready models of GS and AW are given in Table 5. We selected one use case for GS and four use cases for AW. For each use case, we selected the number of elements stereotyped as «BeliefElement» (#Belief), uncertainties (#Uncertainty), known indeterminacy sources (#IndS), known indeterminacy source specifications (#IndSpec), states (#State), and transitions (#Transition). For AW, the percentage of uncertainties specified in the test ready model is more than 50%, which reflects that more than 50% behavior specified in the test ready model is uncertain. This value is higher than the one for GS since the behavior and environment of AW is relatively complex, e.g., a large number of devices.

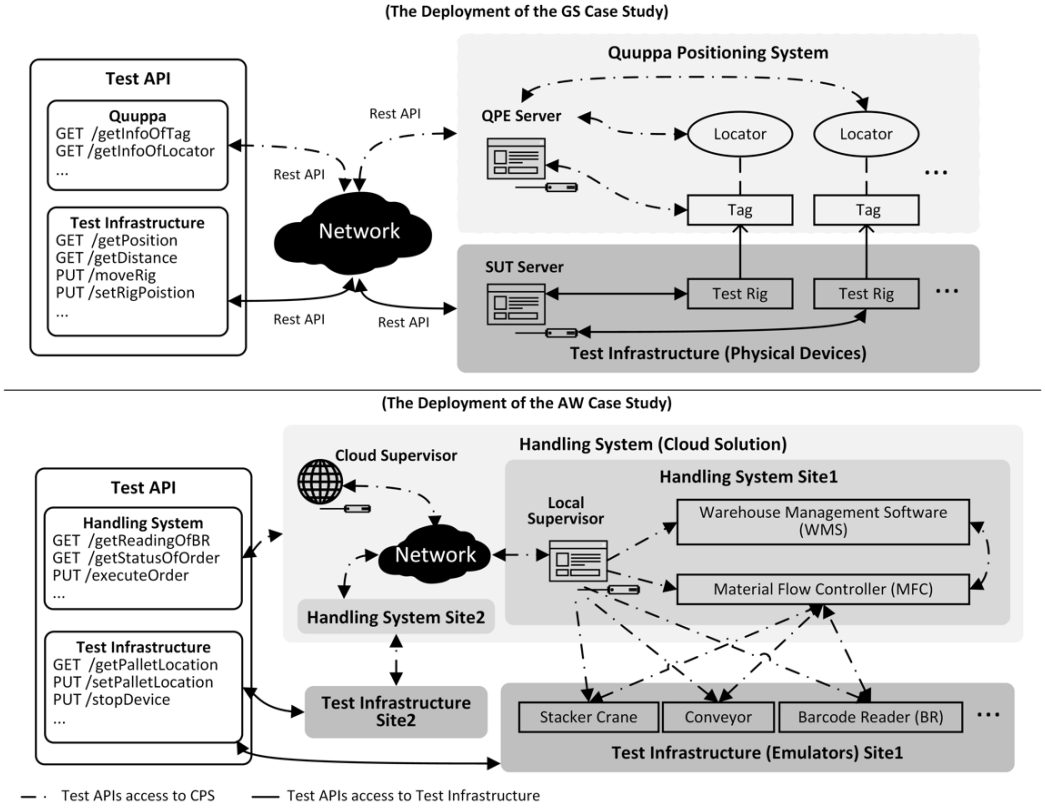


Fig. 13: The Test Execution Solution of the Case Studies

Table 5: Descriptive Statistics of the Case Studies

UC		#Belief	#Uncertainty	#IndS	#IndSp	#State	#Transition	#Uncertainty/#State	#Uncertainty/#Transition
AW	AW1	7	11	2	4	12	15	91.7%	73.3%
	AW2	5	9	2	4	12	18	75.0%	50.0%
	AW3	6	10	-	-	10	14	100.0%	71.4%
	AW4	7	8	1	2	16	16	50.0%	50.0%
GS	GS1	6	6	1	2	17	21	35.3%	28.6%

"-" means unknown indeterminacy source

9.2 Research Questions

We aim to assess which combination of the two test case generation strategies and the four test case minimization strategies is *cost-effective*. In total, we have five combined test strategies. The results for the two test case generation strategies are reported in Table 6. First, test cases are generated from a Belief State Machine using *ASiBP*. With this strategy, the numbers of generated test cases for the two case studies are small, which thus doesn't require test case minimization. The rest of the four strategies are based on test cases generated from a Belief State Machine using *ASlBP*, followed by test case minimization based on the uncertainty related strategies (Section 5.2): average normalized number of uncertainties covered (*Problem 1*), percentage of uncertainty space covered (*Problem 2*), average overall uncertainty measure (*Problem 3*), and percentage of unique uncertainties covered (*Problem 4)*. For simplicity, we refer to these strategies as *Str1* (*ASiBP*), *Str2* (*ASlBP +Problem 1*), *Str3* (*ASlBP + Problem 2*), *Str4* (*ASlBP + Problem 3*) and *Str5* (*ASlBP + Problem 4*) in the rest of the paper. We selected eight commonly used multi-objective search algorithms from the Evolutionary Algorithm, Hybrid Algorithm, and Swarm Algorithm classifications of algorithms. Moreover, we used random search (RS) for the sanity check to determine if complex multi-objective search algorithms are needed, or simply RS suffices.

Table 6: Results of the Test Case Generation Strategies

Case	Use Case	Strategy	#Test Case (nt)	%Transition	%Unique Uncertainty
AW	AW1	*ASiBP*	20	91.3%	100%
		ASlBP	420	100%	100%
	AW2	*ASiBP*	8	88.8%	100%
		ASlBP	776	100%	100%
	AW3	*ASiBP*	5	85.7%	80%
		ASlBP	857	100%	100%
	AW4	*ASiBP*	5	93.7%	100%
		ASlBP	296	100%	100%
GS	GS1	*ASiBP*	5	71.4%	83.3%
		ASlBP	1799	100%	100%

Based on our overall objective, we would like to answer three research questions. **RQ1**: How does the selected multi-objective search algorithms compare to RS regarding solving uncertainty-wise minimization problems (Str2—Str5)? **RQ2**: Which algorithm is the best among selected ones to solve uncertainty-wise minimization problems (Str2—Str5) respectively? **RQ3**: Which uncertainty-wise strategy (Str1-Str5) is effective to discover uncertainties in the real CPS?

9.3 Design of the Evaluation

The design of the evaluation is given in Table 7. The table presents, for each research question, which task we perform, which strategies are compared, which metrics (*Metrics* column) are used, which statistical methods (*Comparison Method* column) are applied, which algorithms are applied, and which case studies are used.

Table 7: Design of the Evaluation

RQ	Experiment Task	Strategy	Metric	Comparison Method	Algorithm		Case Study
1	Compare each algorithm with RS	Str2-Str5	*HV (All), PerTMin (All), AvgNU (Str2), PerUSpace (Str3), AvgUM (Str4), PerUniqueU (Str5), PerTransition (All)*	Vargha and Delaney statistics ($\widehat{A_{12}}$), Kruskal–Wallis Test, Mann-Whitney U Test (p-value)	Evolutionary Algorithm	NSGA-II [36]	AW, GS
						NSGA-III [37]	
						MOCell [38, 39]	
2	Compare each pair of the multi-objective algorithms					SPEA2 [40]	
					Hybrid Algorithm	CellDE [41]	
						AbYSS [41]	
						GDE3 [42]	
					Swarm Algorithm	SMPSO [43]	
					Random Search (only for RQ1)		
3	Compare each pair of the strategies	Str1-Str5	*NumUO, Uk, UkDP,Error, ExeTime, NumTC, EffOT*	Simple Comparison	The best algorithm		

In addition, we provided experiment settings of each strategy regarding test generation and minimization in Table 8. Notice that, to decrease the possibility of obtaining results by chance we ran all the algorithms 100 times for each case study and each strategy [44]. We used the implementation of the eight selected multi-objective search algorithms provided by jMetal [45] with the same number of fitness function evaluations (i.e., 25000) [46] and the following default parameter settings: the *Population Size* of 100, the binary tournament for selecting parents, and the simulated binary criterion for recombination. A crossover rate of 90% was used, and mutation rate was polynomial with the rate of *1.0/n*, where *n* is the number of the bit representation of a solution.

Table 8: Experiment Settings for Each Strategy

Strategy	Generation	Enabling Ind. Source	Minimization with Search		
Str1	*ASiBP*	FindPosition: *Just_Previous* SelectSpecification: *Specified* EnablePattern: *Always*	-		
Str2	*ASIBP* (*max length* is a number of states and transitions in a Belief State Machine)		**Problem.1** *PerTMin, AvgNU, PerTransition*	NSGA-II NSGA-III SPEA2 MOCell CellDE AbYSS GDE3 SMPSO RS	Population size: 100 Max Generation: 25000 Times of Run: 100
Str3			**Problem.2** *PerTMin, PerUSpace, PerTransition*		
Str4			**Problem.3** *PerTMin, AvgUM, PerTransition*		
Str5			**Problem.4** *PerTMin, PerUniqueU, PerTransition*		

To answer RQ1 and RQ2, we compared each pair of the algorithms using HyperVolume (HV) [47] and the individual objectives that are relevant for each strategy. For example, O2 is only valid for Str2. HV was selected based on the guidelines for choosing a quality indicator for search-based software engineering problems that require multi-objective optimization [48]. Based on the guidelines for reporting results for search-based software engineering problems [49], we chose Vargha and Delaney statistics ($\widehat{A_{12}}$) and the Mann Whitney U Test (p-value) to compare the eight selected multi-objective search algorithms with RS for Str2—Str5.

Algorithm 1: Rank Algorithm

```
input: algos[], len(algos)>=2
Output: algos[], rank[]//rank[i] is the rank value of algos[i]
1       n len(algos)
2       for i  1 to n
3          for j  i+1 to n //sort algos[]
4             if better*(algos[i], algos[j])
5                switch(algos,i,j)
6       rank[1]=1;
7       for i  2 to n //set rank values for algos[]
8          if better1(algos[i-1], algos[i])
9             rank[i]=rank[i-1]+1;
10         else
11            rank[i]=rank[i-1];
```

* Function *better(algo1, algo2)* compares *algo1* with *algo2*, which returns the best algorithm based on these two conditions: 1) for HV, p-value<0.05 and $\widehat{A_{12}}$>0.5; 2) p-value<0.05 and $\widehat{A_{12}}$<0.5

Table 9: Definitions of Metrics for Each Research Question

RQ	Metric		Definitions
RQ1 RQ2	$A = \{NSAG-II, NSGA-III, MOCell, SPEA2, CellDE, AbYSS, GDE3, SMPSO, RS\}$, $Str = \{Str2, Str3, Str4, Str5\}$, *Str2*={*PerTMin, AvgNU, PerTransition, HV*}, *Str3*={*PerTMin, PerUSpace, PerTransition, HV*}, *Str4*={*PerTMin, AvgUM, PerTransition, HV*}, *Str5*={*PerTMin, PerUniqueU, PerTransition, HV*}. Note that 1) A_k represents the *k*th *Algorithm*, e.g. A_1 = *NSGA-II*; 2) $Str_{i,j}$ represents the j_{th} objective of the i_{th} strategy, e.g., Str_1 = *Str2*, $Str_{1,1}$ = *PerTMin*.		
	Rank of Algorithm for the objectives of the strategies		$Rank_{A_k}^{Str_{i,j}}$ is the rank value of the A_k algorithm, for the j^{th} objective of Str_i strategy, which is calculated as *rank[k]* in Algorithm 1.
	Confidence of Algorithm for the objectives of the strategies		Confidence of each objective of each strategy is to calculate the percentage of being better than the other algorithms, which is calculated as $Confidence_{A_k}^{Str_{ij}} = (Rank_{A_k}^{Str_{ij}} / \sum_{n=1}^{9} Rank_{A_n}^{Str_{ij}}) \times 100\%$.
	Confidence of Algorithm for the strategies		Confidence of each strategy is to calculate the average confidence of each objective, which is calculated as $Confidence_{A_k}^{S_i} = (\sum_{n=1}^{4} Confidence_{A_k}^{Str_{in}}/4) \times 100\%$.
RQ3	Effectiveness	*NumUO* (*NumUO*$_{Ind}$ *NumUO*$_{ukind}$)	Number of uncertainties occurred during the test set execution, including the occurrence of the uncertainties with the occurrence of their specified indeterminacy sources (*NumUO*$_{Ind}$, i.e., *number of KnOccurred-With-InS of uncertainties*), or unknown indeterminacy sources (*NumUO*$_{ukInd}$, i.e., *number of KnOccurred-Without-InS and number of KnOccurred-UkInS of uncertainties*)
		UO$_{ind}$P	Percentage of times that the introduced indeterminacy sources led to observing corresponding uncertainties during test execution: *UO$_{ind}$P = NumUO$_{Ind}$/ NumUO*
		Error	Number of errors found during a test execution
		Uk	Number of unknown uncertainties occurred during a test set execution (i.e., *number UkOccrred of uncertainties*)
		UkDP	Unknown uncertainty detection percentage: *UkDP = Uk/a number of unique uncertainties.*
	Cost	*ExeTime*	Execution time of the test set
		NumTC	Number of executed test cases
	Efficiency	*EffoT* (*EffoT$_{NUO}$* *EffoT$_{Uk}$*)	*EffoT* represents efficiency in terms of time: 1) *EffoT$_{NUO}$* is the efficiency of uncertainty detection calculated as *NumUO/ExeTime*; 2) *EffoT$_{Uk}$* is the efficiency of unknown uncertainty detection calculated as *Uk/ExeTime.*

Results of test case generation for each case study with each test strategy are represented in Table 6. For S1, the numbers of test cases generated with *ASiBP* for all the case studies were small and didn't require minimization. For Str2 – Str5, we ran each problem 100 times, and thus we combined all the solutions from all the runs for comparison to answer RQ1 and RQ2. To compare the performance of the algorithms, we designed a mechanism to rank all the algorithms based on the $\widehat{A_{12}}$ values and p-values for each metric as shown in the rank algorithm

(Algorithm 1). Furthermore, we calculate the confidence for nine algorithms as shown in Table 10.

For RQ3, we picked the best algorithms for Str2—Str5 based on the results of RQ1 and RQ2, which were used to minimize test cases. The generated test cases for S1 and minimized test cases for Str2 – Str5 were executed on the current deployments of the GS and AW case studies as shown in Fig. 13. The execution results for Str1 – Str5 were evaluated based on various cost, effectiveness, and efficiency measures as shown in Table 9.

9.4 Results and Analyses

9.4.1 Results For RQ1

Recall that RQ1 focuses on comparing the eight selected multi-objective search algorithms with RS based on the individual objectives, HV for (Str2—Str5) minimization problems. Due to a large number of comparisons, the detailed results in terms of rank values, p-values and $\widehat{A_{12}}$ values are provided in submitted supplementary material. The summarized results in terms of confidence and risk (based on the rank of each algorithm) are presented in Table 10 for each case study. For Str2—Str5, for each use case, we can see that RS has the lowest confidence to be the best algorithm. These results suggest that our problems couldn't have been solved effectively with RS and thus the use of complex multi-objective search algorithms is justified.

9.4.2 Results For RQ2

For RQ2, the detailed results of the comparison of each pair of algorithms (C_2^9, i.e., 36 pair-wise comparisons) for each case study for Str2—Str5, in terms of rank values, p-values and $\widehat{A_{12}}$ values are provided in submitted supplementary material. The summarized results in terms of confidence of each algorithm, for each use case is presented in Table 10. As shown in Table 10, in terms of confidence for Str2—Str5, SPEA2 is consistently the best, or the second best (only for two instances of twenty). Based on the results, we recommend using SPEA2 with Str2—Str5 to find the most optimal minimized test cases.

9.4.3 Results For RQ3

To answer RQ3, we chose SPEA2 to minimize test cases for Str2 – Str5 for the two case studies and executed the minimized test cases. The test execution results (together with the execution results for Str1) are provided in Table 11. We compare Str1 – Str5 based on the cost, effectiveness, and efficiency measures (Table 9). In terms of execution time (i.e., a cost measure, presented in column *ExeTime* (s), Table 11), we can observe that Str2 took the highest time to execute for all the use cases except for AW1, where Str4 took the highest time to execute test cases.

In Table 11, the *nt* column shows the number of test cases for each test strategy (Str1 – Str5). Recall from Table 9 that the $UO_{Ind}P$ column shows the percentage of times that the introduced indeterminacy sources led to observing corresponding uncertainties during test execution, whereas the *NumUO* column represents the number of uncertainties that were observed as the result of test execution. As shown in Table 11, consistently for all the five use cases, test cases generated and minimized with Str2 always led to observe more uncertainties when comparing with the others (the *NumUO* column). The $NumUO_{Ind}$ (Table 9) column shows the number of uncertainties out of *NumUO* that occurred because of known indeterminacy sources, whereas the $NumUO_{ukInd}$ column (definition in Table 9) shows the number of uncertainties observed due to unknown indeterminacy sources. Once again Str2 is the best across the case studies in terms of $NumUO_{Ind}$. In terms of $NumUO_{ukInd}$ (except for AW1 where Str4 is the best), Str2 is the best across the case studies. Even for AW1, Str4 observed only one more uncertainty than Str2.

Table 10: Confidence for Each Algorithm for Each Strategy and Each Case Study

Str.	AW1	AW2	AW3	AW4	GS1	Algorithm	AW1	AW2	AW3	AW4	GS1	Str.
Str2	13%	12%	13%	9%	12%	NSGA-II	13%	15%	14%	11%	14%	Str4
	14%	14%	12%	12%	**15%**	NSGA-III	13%	13%	13%	13%	13%	
	8%	8%	8%	9%	9%	MoCell	9%	7%	7%	8%	8%	
	15%	**17%**	16%	**15%**	**15%**	**SPEA2**	**16%**	**17%**	17%	**16%**	**16%**	
	9%	13%	12%	10%	14%	AbYSS	10%	10%	12%	10%	13%	
	8%	5%	7%	8%	7%	CellDE	6%	5%	5%	7%	5%	
	14%	10%	10%	**15%**	10%	GDE3	13%	10%	10%	15%	10%	
	14%	15%	**17%**	14%	12%	SMPSO	15%	16%	**18%**	14%	15%	
	5%	5%	5%	7%	6%	RS	6%	5%	5%	7%	5%	
Str3	13%	13%	13%	12%	11%	NSGA-II	**13%**	13%	13%	11%	12%	Str5
	13%	13%	13%	12%	13%	NSGA-III	**13%**	13%	13%	11%	12%	
	8%	9%	9%	9%	9%	MoCell	8%	9%	9%	9%	9%	
	14%	**15%**	**15%**	**13%**	**15%**	**SPEA2**	**13%**	**15%**	**15%**	**13%**	**15%**	
	10%	12%	12%	11%	14%	AbYSS	10%	12%	12%	12%	13%	
	8%	7%	7%	10%	7%	CellDE	8%	7%	7%	10%	7%	
	12%	10%	10%	**13%**	10%	GDE3	12%	10%	10%	13%	10%	
	14%	13%	13%	12%	14%	SMPSO	**13%**	13%	13%	12%	14%	
	8%	7%	7%	9%	7%	RS	8%	7%	7%	9%	7%	

Table 11: Results for RQ3

UC	Str.	NumTC	Per Transition	ExeTime (s)	$UO_{ind}P$	Num UO	Num UO_{ind}	Num UO_{ukind}	Uk	Err.	UkDP	$EffoT_{NUO}$ /min	$EffoT_{uk}$ /min
AW1	Str1	20	91.3%	216	64 %	25	16	9	10	0	91%	0.116	2.78
	Str2	22	100%	291	64 %	**36**	**23**	13	13	**1**	118%	0.124	2.68
	Str3	17	100%	244	60 %	30	18	12	11	0	100%	0.123	2.70
	Str4	20	96%	519	52 %	29	15	**14**	**16**	**1**	145%	0.056	1.85
	Str5	14	100%	170	59 %	22	13	9	11	0	100%	**0.130**	3.88
AW2	Str1	8	88.8%	387	73 %	11	8	3	0	0	0%	0.028	0
	Str2	106	100%	2134	65 %	**314**	**205**	**109**	**0**	**4**	0%	0.147	0
	Str3	20	100%	866	67 %	52	35	17	0	2	0%	0.060	0
	Str4	54	100%	1114	66 %	148	97	51	0	3	0%	0.133	0
	Str5	30	100%	501	64 %	91	58	33	0	2	0%	**0.182**	0
AW3	Str1	5	85.7%	3156	-	8	-	-	0	0	0%	0.003	0
	Str2	138	100%	99414	-	**955**	-	-	0	**1**	0%	**0.010**	0
	Str3	45	100%	29147	-	271	-	-	0	0	0%	0.009	0
	Str4	92	100%	54990	-	568	-	-	0	**1**	0%	**0.010**	0
	Str5	47	100%	30663	-	305	-	-	0	0	0%	**0.010**	0
AW4	Str1	4	93.7%	8	56 %	9	5	4	0	0	0%	1.089	0
	Str2	24	100%	155	55 %	**296**	**163**	**133**	0	0	0%	1.909	0
	Str3	2	81%	11	48 %	23	11	12	0	0	0%	**2.116**	0
	Str4	7	94%	38	48 %	79	38	41	0	0	0%	2.105	0
	Str5	4	94%	20	53 %	38	20	18	0	0	0%	1.913	0
GS1	Str1	5	71.4%	88	50 %	2	1	1	0	0	0%	0.023	0
	Str2	393	95%	29300	32 %	**1767**	**569**	**1198**	0	0	0%	0.060	0
	Str3	177	100%	12107	29 %	717	211	506	0	0	0%	0.059	0
	Str4	203	100%	12717	31 %	835	259	576	0	0	0%	**0.066**	0
	Str5	174	100%	11428	34 %	715	243	472	0	0	0%	0.063	0

The *Uk* (defined in Table 9) column represents the number of unknown uncertainties observed due to unknown indeterminacy sources. For AW1, with Str4, 16 uncertainties in this category were observed, whereas the second highest was 13 with Str2. The *Error* column represents the number of errors detected with each test strategy. For AW1 and AW2, both Str2 and Str4 observed one error each, whereas, for AW3, Str2 observed four errors, i.e., higher than the other strategies.

Therefore, we recommend Str2 as it performed better than the others in terms of the studied effectiveness measures except for *Uk* and $NumUO_{ukInd}$ for AW1, where Str4 was the second best.

We also compare the strategies based on the efficiency measures. The results are given in the last two columns of Table 11. Note that the efficiency measures tell how many uncertainties (measured with *Uk* and *NumUO*) were observed per minute. For AW1, AW2, and AW3, for the $EffoT_{NUO}/min$ measure, Str5 is the best. For AW4, Str3 is the best with an efficiency value of 2.116 for $EffoT_{NUO}/min$, whereas, for GS, Str4 is the best with an efficiency value of 0.066 *for* $EffoT_{NUO}/min$. However, the differences between these two with the efficiency values of Str5 are not much. For example, for GS, Str5 has an efficiency value of 0.063, i.e., the difference of 0.003 with Str4. This means that Str5 is likely to observe 0.003 fewer uncertainties than Str4 per minutes. Such difference is negligible in practice. In terms of $EffoT_{Uk}/min$ for AW1, once again Str5 is the best strategy. Based on the above results, we suggest using Str2 when the test execution time is not a concern; otherwise, we recommend using Str5 since it is highly likely to be efficient.

9.5 Discussion

In the practice of our two industrial partners, their industrial CPSs were tested with manual test cases. Also, the concept of uncertainty was not at all introduced during the test case development and execution phases of the two industrial partners. In the context of the U-Test project, the two industrial partners provided industrial use cases and implemented the test execution infrastructures (e.g., simulator) for testing their CPSs with UncerTest. We, therefore, tested their CPSs with the test cases generated with UncerTest.

Based on the results and analysis of RQ1, we can conclude that our uncertainty-wise test minimization approaches are complex and thus RS was not sufficient to solve our problems. RS has the lowest confidence to be the best algorithm (i.e., 5.28% on average) as compared to the other algorithms when studying the results of all the use cases together. When comparing the selected multi-objective search algorithms for the four uncertainty-wise test minimization problems (RQ2), we found that SPEA2 has the highest confidence to be the best algorithm (i.e., 12.12% on average) as compared to the other search algorithms and RS.

When comparing the five test strategies, we observed that Str2 (i.e., *ASlBP* with minimization focusing on covering the number of uncertainties) with SPEA2 turned out to be the best. Str2 with SPEA2 observed on average

51%[2] more occurrences of known uncertainties than the other strategies due to unknown indeterminacy sources when combining the results from all the use cases. But Str2 also selects more test cases than the other strategies. In practice, executing more test cases typically requires more resources and therefore increases the time cost. In our context of collaborating with industry, our industrial partners were satisfied with the size of the reduced test set, and therefore they put their focus more on *effectiveness* than *efficiency*. Thus, we recommend Str2 based on this preference of our industrial partners. We also observed that, in one of the five use cases (AW1), in terms of observing unknown uncertainties due to unknown indeterminacy sources, Str4 with SPEA2 performed slightly better than Str2 with SPEA2; Str4 observed 16 unknown uncertainties, but Str2 observed 13. However, more investigation is required to draw any solid conclusion about the performance of Str4 and Str2, which requires conducting more case studies and experiments, which is one of our future plans.

In terms of practical implications, we have four key findings. First, the results of observed known uncertainties due to known indeterminacy sources (the $NumUO_{Ind}$ column) confirm our belief about known uncertainties of the three use cases (AW1, AW2, and AW4) of the AW case study. If the belief is not confirmed (i.e., GS1), it means that the belief of the test modeler about indeterminacy sources is not complete or correct. Then we recommend the test modeler to update her/his belief on indeterminacy sources based on the results of test execution. Second, the results of observed known uncertainties due to unknown indeterminacy sources (the $NumUO_{UkInd}$ column) tell us that the known uncertainties can occur due to the indeterminacy sources that we were not aware of. As a result, such unknown indeterminacy sources need to be investigated and discovered with the help of domain experts in the industry. Once discovered, the test ready models must be updated to reflect these indeterminacy sources. Third, the discovery of unknown uncertainties due to unknown indeterminacy sources (the *Uk* column) need to be investigated once again together with domain experts and reflected in the test ready models as known uncertainties due to known indeterminacy sources (if investigated and found) for future testing. Fourth, the *Error* column tells the errors found during the test execution and must be fixed in the implementation of the CPSs. Note that we observed 15 occurrences of errors for the AW case study. Due to confidentiality issues, further details on the errors and uncertainties cannot be provided. Nonetheless, the results tell us that our proposed test strategies can help us confirming our belief about known uncertainties, discovering unknown uncertainties and unknown indeterminacy sources, and find errors.

9.6 Threats to Validity

External validity. A typical *external validity threat* with any empirical study is related to the generalization of results. Our experiment results were obtained from conducting two industrial case studies (five use cases) from two CPS domains (Automation, Healthcare) and thus additional experiments with different case studies are required to further generalize the results. We would like to point it out that, regarding testing industrial CPSs, it is very expensive as it requires developing test infrastructures. In the context of our project, we luckily had access to the two industrial case studies and the test infrastructures. In the future, we will conduct more industrial case studies to further generalize the results if such opportunities are available.

Internal validity. There are four main internal validity threats in our experiment. First, in terms of test case generation with *ASlBP*, we used the same criteria to generate test cases for all the use cases. This includes generating test cases that must achieve the 100% transition coverage and 100% unique uncertainty coverage. Second, as suggested in [49], all the SBSE problems face the common *internal validity* threat, i.e., parameter settings for the search algorithms. We used the default parameter settings for all the algorithms based on the existing guidelines [49, 50]. Third, we used the same criteria to introduce indeterminacy sources during the test execution for each use case. This means that we used the same values for *EnablePattern*, *FindPosition*, and *SelectSpecification* (Table 8) when executing test cases generated from each test strategy across the use cases. Fourth, the fact that executing each test case more than once can lead to different execution results. Therefore, we executed a test case exactly once if it was included in the test case sets generated by multiple test strategies.

Conclusion validity. There are two main conclusion validity threats in our experiment. First, as discussed in [51], due to randomness in search algorithms, results may have been produced by chance. We handled this threat as suggested in [51], that is to repeat the experiments 100 times. Based on the standard guidelines [49] to report search-based software engineering experiments, we chose the Kruskal–Wallis test to calculate p-value for multiple comparisons with 5% significance level, the Mann-Whitney U test to calculate p-value for pair comparison with 5% significance level, to determine practical and statistical significances of results. Second, our experiment results are based on one-time test execution due to limited resources available to execute test cases on the physical test infrastructures. Additional experiments are required in the future to execute test cases more than once to study whether executing one test case multiple times lead to observing different uncertainties.

Construct validity. As suggested in [44, 52], the same stopping criterion must be used for all the evaluated

[2] The value is calculated as $\frac{\sum_{i=1}^{4}\sum_{j=1,3,4,5}(NumUO_{ukind}^{UC_iStr_2} - NumUO_{ukind}^{UC_iStr_j})/(NumUO_{ukind}^{UC_iStr_2} + NumU)_{ukind}^{UC_iStr_j})}{4\times4}$, where *UC={AW1, AW2, AW4, GS1}, Str = {Str1, Str2, Str3, Str4, Str5}*. $NOU_{ukind}^{UC_1Str_1}$ is the number of uncertainties observed with Str1 for the AW1 use case.

algorithms to avoid any potential bias in results. Following the guidelines, we used the same number of fitness evaluations (25000) and thus dealt with this type of validity threat.

10 Automation

The (open source) tool support[3] for UncerTest is shown in Fig. 14, a user creates a BMs[1] (including Belief Class Diagrams and Belief State Machines) in the IBM Rational Software Architect (RSA) using UncerTum implemented in IBM RSA [9]. In addition to the Belief Class Diagrams and Belief State Machines, the BM also includes object diagrams (of the Belief Class Diagrams), representing test configurations of the CPS being tested.

The first toolset of UncerTest is referred to as *Abstract Test Case Generator*. *AG1* takes Belief State Machines as input and convert them into graphs (SMGraph) in JGraph [53] based on a test case generation strategy (Section 5.3), which can be selected by a tester. *AG2* takes the graph representation of the Belief State Machines as input and converts them into deep paths using the JGrapht tool [53]. Note that multiple regions are not handled by JGrapht, and thus we extended it for this purpose. *AG3* takes the generated deep paths as input and calculates *um* for each path using the *Uncertainty Measurement Calculator* and produces abstract test cases and associated *um* with each test case.

The second toolset is *Uncertainty-Wise Test Case Minimization*. Its *Solution Solver* uses jMetal's implementation of the search algorithms and RS to minimize the number of abstract test cases based on the four test case minimization strategies (Section 6). A tester can select any algorithm and any of the four strategies for test case minimization. The output is a minimized set of test cases and values for the relevant objectives (Section 6). *Solution Processor* converts the output to an EMF model [54], the key input for the third toolset.

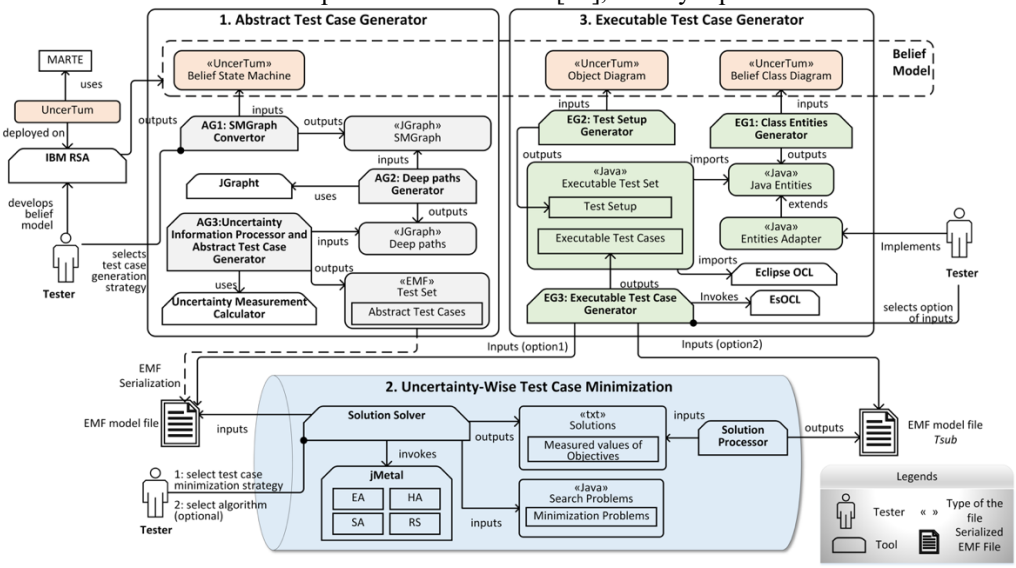


Fig. 14: Automation of UncerTest

The third toolset is *Executable Test Case Generator*. *EG1* takes BCDs as input and converts them to *Java Entities*, which are further extended by a tester as *Entities Adapter* to provide actual implementation of operations, e.g., how to invoke REST APIs in GS. For each case study, a user has to manually implement *Entities Adapters* to bridge the gap between model elements and implementation of Test API. *EG2* takes the object diagram as input and outputs *Test Setup*, which is required for the execution of test cases. Finally, *EG3* takes the EMF model file as the input and invokes EsOCL [30] to obtain concrete test data. EsOCL is a search-based OCL solver that takes input an OCL constraint and provides a set of data that satisfies the constraint. Using the output from EsOCL, EG3 produces executable test cases, where each executable test case imports Eclipse OCL [54] to check OCL constraints (state invariants) at runtime, which serve as test oracles.

11 Related Work

Walkinshaw and Fraser [5] proposed a black-box testing framework to select test cases for execution to decrease uncertainty about the correctness of a software system. The framework relies on Genetic Programming (GP) [55] to infer models of a system under test. It generates random inputs and assesses them on the inferred models to select ones that create most uncertainty, and eventually only execute the selected ones on the real system under test. Uncertainty was measured as the level of confidence in the corresponding output of input (i.e., test data). UncerTest shares a similar objective, that is, selecting test cases for execution by taking into account uncertainty. Differences between the two approaches can be summarized from the three aspects: 1) UncerTest focuses on

[3] The tool for UncerTest is open source, which is available at https://bitbucket.org/ManZH/uncertest-v1.

testing CPS under uncertainty, but their proposed framework is for software; 2) UncerTest requires initial BMs with subjective uncertainty specified as the input, whereas in their approach models are inferred by GP, which requires the execution of the software under test; and 3) UncerTest elaborates uncertainty from the aspects of number of uncertainties, number of unique uncertainties, uncertainty space, and uncertainty measure from the Uncertainty Theory, whereas their approach is based on an existing uncertainty sampling technique.

Iqbal et al. [56] proposed an environment modeling approach that handles uncertainty to support automated testing. The modeling of the following uncertain situations is supported: 1) Uncertainty on a value: Instead of specifying an exact value, the modeling approach supports specifying a range of possible values, i.e., upper and lower bounds; 2) Uncertainty on transitions: This allows attaching probability to a transition indicating the probability with which a state can be reached from another one with/without specific events. In addition, a time uncertainty can be further attached to a transition with a specific probability, indicating the probability of timeout of the transition; 3) Uncertainty on scenarios: The approach also provides a solution to model a situation in a UML state machine, where an event may lead to uncertain outcomes, by specifying a choice node that can reach multiple target states without any guards. Based on such models, the approach enables to generate corresponding environment simulators with configurations, interacting with a system under test. Also, Iqbal et al. [56] designed search heuristics for seeking optimal simulation configurations that can lead the environment into error states. By comparing with UncerTest, uncertainties in their approaches are considered in environmental context, and UncerTum [9, 10] (UncerTest's associated modeling solution) supports to model uncertainty regarding a system and its environment. In other words, their approach focuses on modeling possible environment situations that the system may face during operation, and our approach focuses on modeling possible uncertain behaviors of the system with the linked known indeterminate environment. Our modeling support, i.e., UncerTum [9, 10] provides a comprehensive capability of modeling uncertainty. We can cover all situations that they modeled by applying the UML Uncertainty Profile (UUP) [9, 10]. Also, we explicitly handle types of uncertainties, e.g., uncertainty on a value can be considered as a content uncertainty and uncertainty modeled in state machines (i.e., uncertainty in scenarios described above) can be considered as occurrence uncertainty together with time uncertainty. Furthermore, we provided advance modeling features for uncertainty measurement (e.g., fuzzy set [57], belief interval [58]), which is not limited to simple ranges or probabilities as in their approach. In addition, our approach with search aims at identifying specified uncertainties and discovering unknown uncertainties by considering different perspectives of uncertainties (e.g., measurement, number of uncertainties covered by test cases). In their approach, the search is used for obtaining environment simulation configurations that can lead the environment into known errors with test data generated with the EsOCL OCL solver [59].

Another related work [6] focuses exclusively on time-related uncertainty. It relies on UML sequence diagrams together with the UML Profile for Schedulability, Performance, and Time (SPT) [60]. This work, however, only supports modeling uncertainty in time on messages of sequence diagrams. As discussed in Section 2.1, UncerTest is built on *UncerTum* [9], which is a comprehensive modeling framework for specifying various types of uncertainty (e.g., time, content and environment). The work presented in [6] focuses on stress testing of systems in the existence of time-related uncertainty on messages, which may complement the UncerTest framework, which can be investigated in the future.

David et al. [61] presented some test generation principles and algorithms (e.g., the online testing tool UPPAAL-TRON [62]) and discussed the feasibility of applying them for testing timed systems under uncertainty, at a high level of abstraction. In their context, uncertainty is caused by the inherent concurrent and indeterminate nature of timed systems. UncerTest, however, addresses uncertainty with a much broader scope and has an end-to-end MBT solution.

To model uncertainty (inherent in real-world applications) with UML class diagrams, an extension was proposed in [63-65], which is referred to as fuzzy UML data modeling. The extension relies on two theories: fuzzy set and possibility distribution, and was later on further extended in [66] to transform fuzzy UML data models into representations in the fuzzy description logic (FDLR) to check the correctness of fuzzy properties. Furthermore, another automated transformation was proposed in [67] to transform fuzzy UML data models into web ontologies to support automated reasoning on fuzzy properties in the context of web services. These works focus on the analyses at the design time, whereas our work focuses on testing. Regarding modeling, our *UncerTum* focuses on uncertainty in a comprehensive and precise manner by considering various types of measures such as probability, vagueness, and fuzziness. The methodologies proposed in [63-65] for specifying fuzzy UML data can easily integrate with our model libraries when needed and potentially used to support MBT of CPSs under uncertainty. However, this requires further investigation.

There exist several works [68-71] that use usage models with associated probability information for statistical testing. Usage models represent excepted behaviors based on the actual use of the software. For instance, Prowell [68] proposed a tool, named as JUMBL, which uses Markov chains as usage models for supporting test case generation and system reliability analysis. The probability information can be obtained by relying on, e.g., usage profiles of software or domain expertise. First, these works consider the subjective perspective due to the application of usage models based on domain expertise, which is the same with UncerTest. However, these works

are based on the probability theory (frequency), whereas UncerTest is based on the uncertainty theory (belief degree) [17]. Second, these works intend testing a system by detecting defects that lead to system failures with higher frequencies. However, the *Str4* (*ASlBP* + AUM) of UncerTest tests a CPS by identifying uncertainties of higher belief degrees. In addition, UncerTest provided other four strategies, which support broader applications, e.g., when measurements are not accessible.

In [72], a language-independent solution was proposed consisting of partiality, *Abs* partiality, *Var* partiality, and *OW* partiality, to denote the degree of incompleteness specified by model designers. The work also provides a solution for merging and reasoning possible partial models with tool support [72, 73]. The approach was demonstrated in UML class and sequence diagrams [72]. This work is related to our work regarding expressing the uncertainty of modelers. However, in the context of their work, the focus is on uncertainty in partial models for supporting model refinement and evolution. In contrast, we focus on modeling uncertainty (lack of confidence) in test ready models that are used for test case generation and minimization relying on the uncertainty theory.

12 Conclusion and Future work

Nowadays, Cyber-Physical Systems (CPSs) are everywhere in our daily life. It is forecasted that applications of CPSs will span over many different domains shortly, including autonomous vehicles, robotics, healthcare, industrial automation, among others. One critical dimension of the complexity of developing and testing such systems is due to the inherent uncertainty of their operational environment and uncertain behaviors of themselves. To tackle this challenge, in this paper, we proposed a model-based and search-based test case generation and minimization framework (named as UncerTest) for testing CPSs under uncertainty. UncerTest takes advantages of the uncertainty theory and search-based optimization techniques, based on which, it also proposes an innovative set of uncertainty-related test case minimization strategies. We evaluated UncerTest with two industrial CPSs case studies and eight commonly used multi-objective search algorithms. The best test strategy managed to discover on average 51% more uncertainties due to unknown indeterminacy sources as compared to the rest of the test strategies across the case studies. The same test strategy managed to discover 118% more unknown uncertainties as compared to the already known ones.

In the future, we plan to conduct additional experiments (e.g., executing test cases multiple times) with more case studies, and further study correlations between the uncertainty-related objectives (e.g., *um*) and the identification of unknown uncertainties. Moreover, we only applied one strategy to introduce indeterminacy sources, and there is a need to develop different strategies and evaluate their performance. We also plan to consider specifications of the introducing indeterminacy source (e.g., measurement of indeterminacy source and a position to be introduced) as factors to guide how to generate executable test case using multi-objective search algorithms.

Acknowledgment

This research was initially supported by the EU Horizon 2020 funded project U-Test (Testing Cyber-Physical Systems under Uncertainty, Project Number: 645463). The research, however, was finalized after the project with the support from RCN funded MBT4CPS project, who funded Man Zhang. Tao Yue and Shaukat Ali are also supported by the Research Council of Norway funded Zen-Configurator project. Tao Yue is also supported by the National Nature Science Foundation of China 61872182. Man Zhang is also funded by the Research Council of Norway funded EET project (Evolutionary Enterprise Testing). The corresponding author of the paper is Tao Yue. We sincerely thank our industrial partners (ULMA Handling Systems and Nordic Medtest), especially Oscar Okariz and Malin Hedman, for their support on providing the case studies.

References

[1] D. B. Rawat, J. J. Rodrigues, and I. Stojmenovic, *Cyber-physical systems: from theory to practice*: CRC Press, 2015.

[2] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling Cyber-Physical Systems," *Proceedings of the IEEE,* vol. 100, no. 1, pp. 13-28, 2012.

[3] M. Woehrle, K. Lampka, and L. Thiele, "Conformance testing for cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)* vol. 11, no. 4, pp. 1-23, 2013.

[4] H. Abbas, B. Hoxha, G. Fainekos, J. V. Deshmukh, J. Kapinski, and K. Ueda, "Conformance testing as falsification for cyber-physical systems," *arXiv preprint arXiv:1401.5200*, 2014.

[5] N. Walkinshaw, and G. Fraser, "Uncertainty-Driven Black-Box Test Data Generation." pp. 253-263.

[6] V. Garousi, "Traffic-aware stress testing of distributed real-time systems based on UML models in the presence of time uncertainty." pp. 92-101.

[7] G. Bammer, and M. Smithson, *Uncertainty and risk: multidisciplinary perspectives*: Routledge, 2012.

[8] D. V. Lindley, *Understanding uncertainty (revised edition)*: John Wiley & Sons, 2014.

[9] M. Zhang, S. Ali, T. Yue, and R. Norgre, *An Integrated Modeling Framework to Facilitate Model-Based Testing of Cyber-Physical Systems under Uncertainty,* Technical report 2016-02, Simula Research Laboratory, 2016.

[10] M. Zhang, S. Ali, T. Yue, R. Norgren, and O. Okariz, "Uncertainty-Wise Cyber-Physical System test modeling," *Software & Systems Modeling*, 2017/07/25, 2017.

[11] F. P. X. "Future Position X," 2017; http://www.fpx.se/.

[12] ULMA. "ULMA Handling System," 2017; http://www.ulmahandling.com/en/.

[13] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding Uncertainty in Cyber-Physical Systems: A Conceptual Model." pp. 247-264.

[14] OMG, "UML Profile For MARTE: Modeling And Analysis Of Real-Time Embeded Systems™," 2011.

[15] Y. Li, J. Chen, and L. Feng, "Dealing with uncertainty: a survey of theories and practices," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 25, no. 11, pp. 2463-2482, 2013.

[16] B. Liu, "Why is there a need for uncertainty theory," *Journal of Uncertain Systems,* vol. 6, no. 1, pp. 3-10, 2012.

[17] B. Liu, *Uncertainty theory*: Springer, 2015.

[18] Y. Zhu, "UNCERTAIN OPTIMAL CONTROL WITH APPLICATION TO A PORTFOLIO SELECTION MODEL," *Cybernetics and Systems,* vol. 41, no. 7, pp. 535-547, 2010/09/24, 2010.

[19] L. Yang, K. Li, and Z. Gao, "Train Timetable Problem on a Single-Line Railway With Fuzzy Passenger Demand," *IEEE Transactions on Fuzzy Systems,* vol. 17, no. 3, pp. 617-629, 2009.

[20] J. Peng, "Risk metrics of loss function for uncertain system," *Fuzzy Optimization and Decision Making,* vol. 12, no. 1, pp. 53-64, 2013//, 2013.

[21] S. Han, Z. Peng, and S. Wang, "The maximum flow problem of uncertain network," *Information Sciences,* vol. 265, pp. 167-175, 5/1/, 2014.

[22] R. S. Pressman, *Software engineering: a practitioner's approach 7th edition*: Palgrave Macmillan, 2010.

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

[24] W. Rudin, *Real and complex analysis*: Tata McGraw-Hill Education, 1987.

[25] J. Offutt, and A. Abdurazik, "Generating tests from UML specifications." pp. 416-429.

[26] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, "Generating test data from state-based specifications," *Software testing, verification and reliability,* vol. 13, no. 1, pp. 25-53, 2003.

[27] P. Samuel, R. Mall, and A. K. Bothra, "Automatic test case generation using unified modeling language (UML) state diagrams," *IET software,* vol. 2, no. 2, pp. 79-93, 2008.

[28] L. C. Briand, Y. Labiche, and Y. Wang, "Using simulation to empirically investigate test coverage criteria based on statechart." pp. 86-95.

[29] OMG, "Object Constraint Language™ (OCL™)," 2014.

[30] S. Ali, M. Z. Iqbal, A. Arcuri, and L. C. Briand, "Generating test data from OCL constraints with search techniques," *IEEE Transactions on Software Engineering,* vol. 39, no. 10, pp. 1376-1402, 2013.

[31] OMG, "UML Testing Profile (UTP™) 1.2," 2013.

[32] Quuppa. "Quuppa - Do more with Location," 2017; http://quuppa.com/.

[33] N. M. Test. "Nordic Med Test," 2017; http://www.nordicmedtest.se/.

[34] IK4-IKERLAN. "IK4-IKERLAN," 2017; http://www.ikerlan.es/eu/.

[35] U-Test. "Use Cases - Industrial Case Studies," 2017; http://www.u-test.eu/use-cases/.

[36] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation,* vol. 6, no. 2, pp. 182-197, 2002.

[37] K. Deb, and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Transactions on Evolutionary Computation,* vol. 18, no. 4, pp. 577-601, 2014.

[38] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, “Mocell: A cellular genetic algorithm for multiobjective optimization,” *International Journal of Intelligent Systems,* vol. 24, no. 7, pp. 726-746, 2009.
[39] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "Design issues in a multiobjective cellular genetic algorithm." pp. 126-140.
[40] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," *International Center for Numerical Methods in Engineering*.
[41] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham, “AbYSS: Adapting scatter search to multiobjective optimization,” *IEEE Transactions on Evolutionary Computation,* vol. 12, no. 4, pp. 439-457, 2008.
[42] S. Kukkonen, and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution." pp. 443-450.
[43] A. J. Nebro, J. J. Durillo, J. Garcia-Nieto, C. A. C. Coello, F. Luna, and E. Alba, "SMPSO: A new pso-based metaheuristic for multi-objective optimization." pp. 66-73.
[44] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, “A systematic review of the application and empirical investigation of search-based test case generation,” *IEEE Transactions on Software Engineering,* vol. 36, no. 6, pp. 742-762, 2010.
[45] A. J. Nebro, and J. J. Durillo. "jMetal," 2016; http://jmetal.sourceforge.net/.
[46] M. Črepinšek, S.-H. Liu, and M. Mernik, “Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them,” *Applied Soft Computing,* vol. 19, pp. 161-170, 2014/06/01/, 2014.
[47] E. Zitzler, and L. Thiele, “Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach,” *IEEE transactions on Evolutionary Computation,* vol. 3, no. 4, pp. 257-271, 1999.
[48] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen, "A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering." pp. 631-642.
[49] A. Arcuri, and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering." pp. 1-10.
[50] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*: CRC Press, 2003.
[51] S. Wang, S. Ali, and A. Gotlieb, "Minimizing test suites in software product lines using weight-based genetic algorithms." pp. 1493-1500.
[52] M. de Oliveira Barros, and A. C. Dias-Neto, *Threats to Validity in Search-based Software Engineering Empirical Studies,* Technical Report 0006/2011, Universidade Federal Do Estado Do Rio de Janeiro, 2011.
[53] B. Naveh. "JGrapht," 2016; http://jgrapht.org/.
[54] E. MDT. "Eclipse OCL," 2016; http://www.eclipse.org/modeling/mdt/?project=ocl#ocl.
[55] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*: Lulu. com, 2008.
[56] M. Z. Iqbal, A. Arcuri, and L. Briand, “Environment modeling and simulation for automated testing of soft real-time embedded software,” *Software & Systems Modeling,* vol. 14, no. 1, pp. 483-524, 2015/02/01, 2015.
[57] L. A. Zadeh, “Fuzzy sets,” *Information and control,* vol. 8, no. 3, pp. 338-353, 1965.
[58] A. P. Dempster, “Upper and lower probabilities induced by a multivalued mapping,” *The annals of mathematical statistics*, pp. 325-339, 1967.
[59] S. Ali, M. Z. Iqbal, A. Arcuri, and L. Briand, "A Search-Based OCL Constraint Solver for Model-Based Test Data Generation." pp. 41-50.
[60] OMG, "UML Profile For Schedulability, Performance, and Time™," 2005.
[61] A. David, K. G. Larsen, S. Li, M. Mikucionis, and B. Nielsen, "Testing real-time systems under uncertainty." pp. 352-371.
[62] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, and A. Skou, "Testing real-time systems using UPPAAL," *Formal methods and testing*, pp. 77-117: Springer, 2008.
[63] Z. Ma, “Fuzzy information modeling with the UML,” *Idea*, 2005.
[64] Z. M. Ma, F. Zhang, and L. Yan, “Fuzzy information modeling in UML class diagram and relational database models,” *Applied Soft Computing,* vol. 11, no. 6, pp. 4236-4245, 2011.
[65] L. Yan, and Z. M. Ma, "Extending nested relational model for fuzzy information modeling." pp. 587-590.
[66] Z. M. Ma, F. Zhang, L. Yan, and J. Cheng, “Representing and reasoning on fuzzy UML models: A description logic approach,” *Expert Systems with Applications,* vol. 38, no. 3, pp. 2536-2549, 2011.
[67] F. Zhang, and Z. M. Ma, “Construction of fuzzy ontologies from fuzzy UML models,” *International Journal of Computational Intelligence Systems,* vol. 6, no. 3, pp. 442-472, 2013.
[68] S. J. Prowell, "JUMBL: a tool for model-based statistical testing." p. 9 pp.
[69] M. Riebisch, I. Philippow, and M. Götze, "UML-based statistical test case generation," *Objects, Components, Architectures, Services, and Applications for a Networked World*, pp. 394-411: Springer, 2002.
[70] G. H. Walton, J. H. Poore, and C. J. Trammell, “Statistical testing of software based on a usage model,” *Software: Practice and Experience,* vol. 25, no. 1, pp. 97-108, 1995.

[71] D. P. Kelly, and R. S. Oshana, “Improving software quality using statistical testing techniques,” *Information and Software Technology,* vol. 42, no. 12, pp. 801-807, 2000.
[72] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty." pp. 573-583.
[73] M. Famelis, and S. Santosa, "MAV-Vis: a notation for model uncertainty." pp. 7-12.

Appendix A. List of Abbreviations

Abbreviations	Description
$u(s_x, t_z, s_y)$	An uncertainty in a Belief State Machine is denoted as $u(s_x, t_z, s_y)$, s_x is a source state, s_y is a possible target state transited by a transition t_z from s_x.
$usp(s_x, t_z)$	An Uncertainty Space in a Belief State Machine is denoted as $usp(s_x, t_z)$, which represents a set of uncertainties that originate from same source state s_x by same transition t_z. $usp(s_x, t_z) = \{u_i \mid i = 1...n\}$.
um(x)	*um(x)* denotes an uncertainty measurement of e.g., an uncertainty, a test case. Theoretically, a measurement of an *usp* is always 1 or Certain.
ASiBP	All Simple Belief Path Coverage, i.e., a coverage criteria to generate test cases
ASlBP	All Specified Length Belief Path Coverage, i.e., a coverage criteria to generate test cases
PerTMin	Percentage of Test Case Minimization, i.e., one of test minimization objectives for reducing a number of test cases
AvgNU	Average Normalized Number of Uncertainties Covered, i.e., one of test minimization objectives for searching an optimal set of test cases with more uncertainties
PerUSpace	Percentage of Uncertainty Space Covered, i.e., one of test minimization objectives for searching an optimal set of test cases with more uncertainty spaces.
AvgUM	Average Overall Uncertainty Measure, i.e., one of test minimization objectives for searching an optimal set of test cases with higher uncertainty measurement
PerUniqueU	Percentage of Unique Uncertainties Covered, i.e., one of test minimization objectives for searching an optimal set of test cases with more unique uncertainties
PerTransition	Percentage of Transition Coverage, i.e., one of test minimization objectives for searching an optimal set of test cases with a higher coverage of a test ready models