Testing Self-Adaptive Software with Probabilistic Guarantees on Performance Metrics: Extended and Comparative Results

Claudio Mandrioli, and Martina Maggio

Abstract—This paper discusses methods to test the performance of the adaptation layer in a self-adaptive system. The problem is notoriously hard, due to the high degree of uncertainty and variability inherent in an adaptive software application. In particular, providing any type of formal guarantee for this problem is extremely difficult. In this paper we propose the use of a rigorous probabilistic approach to overcome the mentioned difficulties and provide probabilistic guarantees on the software performance. We describe the set up needed for the application of a probabilistic approach. We then discuss the traditional tools from statistics that could be applied to analyse the results, highlighting their limitations and motivating why they are unsuitable for the given problem. We propose the use of a novel tool – *the Scenario Theory* – to overcome said limitations. We conclude the paper with a thorough empirical evaluation of the proposed approach, using three adaptive software applications: the Tele-Assistance Service, the Self-Adaptive Video Encoder, and the Traffic Reconfiguration via Adaptive Participatory Planning. With the first, we empirically expose the trade-off between data collection and confidence in the testing campaign. With the second, we demonstrate how to compare different adaptation strategies. With the third, we discuss the role of the randomisation in the selection of test inputs. In the evaluation, we apply the scenario theory and also classical statistical tools: Monte Carlo and Extreme Value Theory. We provide a complete evaluation and a thorough comparison of the confidence and guarantees that can be given with all the approaches.

Index Terms—Testing, Self-Adaptive Software, Autonomous Systems.

1 Introduction

Software systems are affected by uncertainty that alters their behaviour and can render their performance unpredictable. Adaptation layers were introduced in software as a viable solution to deal with performance fluctuations and minimise the effect of uncontrolled changes [1], [2], [3]. This makes software self-adaptive. The idea behind self-adaptive software is to have a layer responsible for observing behavioural changes and taking counteractions. This can guarantee more stable and predictable software performance in terms of non-functional software behaviour [4], [5], e.g., lower response times, or higher reliability.

Adaptation can be implemented using different methodologies; some of them provide guarantees based on formal models [5], [6], [7], others are empirically proven effective [8], [9], [10]. In both cases there is a need for appropriate performance testing of the system composed of the software and its adaptation layer. The presence of an adaptation layer opens up the possibility that in the same exact condition the software will behave differently, depending on its past behaviour and accumulated knowledge. It is necessary to conduct empirical validation of satisfactory behaviour to verify the correctness of the system and adaptation-layer implementation [11]. In addition, it is important to quantify the achievable performance.

- *C. Mandrioli is a Ph.D. student at the Department of Automatic Control, Lund University, Sweden.*
  E-mail: claudio.mandrioli@control.lth.se
- *M. Maggio is with the Department of Computer Science at Saarland University, Germany and with the Department of Automatic Control at Lund University, Sweden.*
  E-mail: maggio@cs.uni-saarland.de

Manuscript received September XX, XX; revised XX.

In general, testing is a crucial aspect of software development. For self-adaptive software, the testing process is complicated by the presence of the adaptation layer [12], [13], [14], [15]. Self-adaptive systems testing is intrinsically hard, due to the extreme variability and uncertainty involved in the software execution [14], [16], [17], [18]. In fact, the adaptation layer explicitly reacts to the uncertainty, and may influence it for the future. This creates a *loop* around the software [1]. In the context of uncertainty and adaptation, this paper's challenge is to achieve and maintain formal guarantees on non-functional aspects of the software execution, such as reliability and response times.

Research Challenges: The adaptation layer and the presence of uncertainty impose specific challenges for testing. Triggered by the environmental variability, the adaptation generates changes in the system - and this changing nature makes it difficult (and in many cases impossible) to *exhaustively* guarantee its correct behaviour [14], [19], [20], [21], [22], [23]. The adaptation also creates a difficulty in the performance quantification and in determining the testing sufficiency and effectiveness [14], [16].

As an example, consider testing a web-application that can run on different servers with different and time-varying performance results. Not all of the servers can provide the same reliability. The adaptive layer should choose dynamically which server to use, in order to maximise the overall reliability. In general, it is not possible to guarantee that the application is always reliable, since any server may fail. Also, the actual reliability will depend significantly

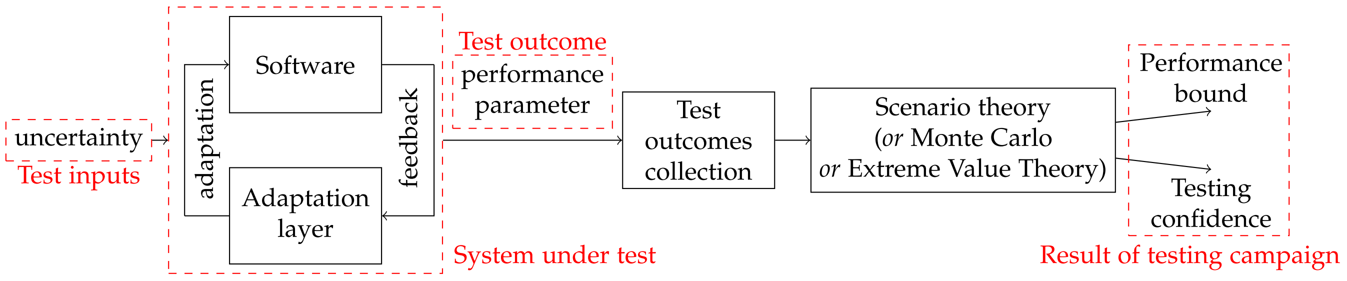

Fig. 1. Overview of the proposed approach. In the figure, black blocks represent components of the adaptive system and of the testing procedure. The red text and boxes highlight the main high-level concepts of the testing procedure. The arrows represent information flow. Figure from [24].

on the specific servers, and on their performance. As a consequence, when testing the system, any evaluation of its reliability is heavily affected by the specific test cases. Determining which tests are sufficient and when it is possible to stop the testing process becomes challenging. Prior literature contributions highlighted a set of research challenges for testing adaptive software [14], [16]. The quoted papers listed a series of challenges, and then group them into macro areas: types of guarantees, quantification of performance metrics, and quantification of the testing effectiveness. In this paper, we try to address these macro challenges:

- **CH1**: Definition of what type of guarantees can be given for self-adaptive software.
- **CH2**: Quantification of the mentioned guarantees.
- **CH3**: Quantification of the testing sufficiency and effectiveness (or testing adequacy).

Furthermore, any method based on statistics heavily depends on the input data. In this paper we also discuss an additional challenge (that is mentioned in [14], [16] with less emphasis):

- **CH4**: Definition and collection of testing input data.

We believe **CH1** to be the main challenge that we try to solve in this paper. The remaining challenges can be viewed as sub-challenges, that mark relevant aspects in the definition of a methodology to address CH1. Towards a solution for **CH1**, we define a particular notion of probabilistic guarantees that can be provided after a testing phase of self-adaptive strategies and software. The need to provide such formal guarantees, forces us to think and consider the other relevant aspects. Probabilistic guarantees trigger the need for repeated testing, resulting in the need of generating randomised test cases and to carefully select input data, therefore triggering **CH4**. When tests are performed, the data gathered in the testing phase should be analysed to quantify the provided guarantees, giving an answer to **CH2**. It is also important to quantify the testing adequacy – as mentioned in **CH3** – i.e. to understand if the number of conducted tests is sufficient or if more tests are needed. This clearly depends on the nature and type of the probabilistic guarantees to be produced.

Contribution: In this paper we address the four mentioned research challenges by leveraging a rigorous probabilistic approach [25], [26]. The probabilistic approach is beneficial in two ways: (i) it allows the efficient exploration of large input and configuration spaces [27], and (ii) it can provide a quantification of its own adequacy. In the field of probability theory, the testing adequacy is called *confidence*.

As discussed in the research challenges above, the uncertain nature of self-adaptive systems does not allow for the definition of strict guarantees. This limitation mainly arises from the large (and possibly infinite) number of combinations of inputs that can be provided to the system [14], [16]. Despite this, *we need to test self-adaptive systems when said variability is present, in order to trigger the adaptive behaviour*. Leveraging a probabilistic approach, the uncertainty and variability can efficiently be explored using randomised inputs. As a consequence, the measured performance metric must be treated as a random quantity, and requires statistical evaluation. We therefore enter the domain of probabilistic guarantees [25], [28]. The randomised approach allows for an efficient exploration that is independent from the size and quantity of the uncertainty that is present in the software execution [26], [27].

In this work, we focus on the evaluation of probabilistic bounds for a given performance metric. In contrast with *conventional* testing, *statistical* testing can only provide confidence values. In conventional testing, a property is either evaluated to true or false. When the same test is repeated with a different set of random input values, the property evaluation becomes a stochastic variable. This means that the result of the testing process is the confidence in the property being either true or false, when repeating the test with a new input set.

According to the probabilistic framework, our aim is testing what is the value of the performance parameter that the adaptive software can guarantee in the "majority" of its execution environments. We formally define majority in a probabilistic fashion, e.g., that a given performance bound will hold in 99% of the execution instances. We also quantify the confidence that we can claim, i.e., the adequacy of our testing campaign. High confidence means a high probability that we performed a sufficient number of randomly generated tests to sustain our claim. In some sense, this is analogous to a *coverage criterion* – a reference for choosing when to stop the testing campaign. One of the contributions of the paper is the discussion of input data. We provide guidelines on what needs to be randomised and what – in contrasts – remains fixed over the set of executed tests. We also discuss testing a partial system in contrast to testing the real system in operation, where there is no need for randomisation of input data because the real execution uses a *realistic* set of execution conditions.

In the paper, we discuss traditional tools from statistics (Monte Carlo and Extreme Value Theory) and highlight their limitations for testing self-adaptive software. We overcome these limitations using a tool called *Scenario Theory* [29]. The Scenario Theory was developed in the field of robust control but can actually be applied to a very general class of problems. In this paper we show how to apply it to the problem of testing self-adaptive software. We provide a thorough comparison between the confidence and the results obtained with the Scenario Theory and with Monte Carlo and Extreme Value Theory.

Experimental Evaluation: To support our claims, we use our methodology to test the behaviour of three self-adaptive software applications: the Tele-Assistance System [30], the Self-Adaptive Video Encoder [31], and the Traffic Reconfiguration via Adaptive Participatory Planning [32]. We show the complete application of Monte Carlo, Extreme Value Theory and the Scenario Theory. In all cases, we discuss how these methods can be used to: (i) rigorously quantify the adaptation performance, (ii) evaluate the trade-off between the number of performed tests and the confidence in the testing campaign, and (iii) compare adaptation strategies.[1] Our experimental results show that the Scenario Theory provides better guarantees and higher confidence in the results.

Extension: This paper extends our previous work [24] providing novel contribution, both on the methodological discussion, and on the empirical evaluation. For what concerns the methodology, we present a detailed discussion of the application of traditional statistical tools to the testing of self-adaptive systems. We also apply the mentioned tools in our empirical evaluation, comparing the results obtained with our proposed testing strategy. We included an additional case study to evaluate the methodology on a different software application, in particular with respect to the relevance of the randomisation of the input data.

Paper Structure: In Section 2 we provide an overview of our proposed testing approach. Section 3 discusses related work. Section 4 presents our methodology and describes how it overcomes the limitations of classical statistical testing. Section 5 presents experimental results. Finally, Section 6 discusses the limitations and threats to validity of the proposed approach and Section 7 concludes the paper.

2 Approach Overview

In this section we provide an overview of our testing approach (shown in Figure 1). In particular, we discuss: (i) the definition of *test inputs*, (ii) the definition of the *test outcome*, and (iii) the evaluation of the *results of the testing campaign*. In Sections 3.2 and 4 we discuss in detail how to apply respectively traditional tools and the Scenario Theory to evaluate the test outcomes and obtain *performance bounds* and *testing confidence*.

The objective of our testing campaign is to empirically provide guarantees on the system behaviour. These guarantees should be general and independent of the specific test cases. Practically, we want independence from the variability and uncertainty that affects the executed tests. We obtain this by performing different random tests, each of which represents a possible system realisation. We then statistically evaluate the results of the testing campaign.

Performing repeated random tests requires, as a first step, the definition of what are the test inputs that need randomisation. The remaining inputs, instead, should be fixed across the tests. Using the web application example, we can say that the number of connected users is a parameter that should vary from one test to the next, but (possibly) the amount of threads that are assigned to serving requests from clients is fixed and defined by the specification of the software architecture.

The choice of what to randomise and what to keep constant highly impacts the significance of the testing campaign. If more than necessary test inputs are randomised, the testing results can be unnecessarily conservative. Conversely, fixing inputs that will actually vary in the actual software execution – and therefore are uncertain and unknown – will provide results that do not carry on between the testing campaign and the actual software implementation.

The testing engineer should randomise all inputs that are not known at development time and that will affect the system performance. Most importantly, these have to include the exogenous inputs that the software is supposed to adapt to. In this way, the random sampling will trigger the adaptation layer and explore the possible performances of the system. From a practical point of view, the randomised inputs should be all of those inputs that will make the same implementation of the system potentially behave at a different performance level. The definition and analysis of the relevant test inputs are domain and application dependant: different types of adaptive software are required to respond to different inputs (e.g. discrete signals or continuous quantities). Moreover, an evaluation on the quality of the test inputs requires assumptions and modeling efforts so that an analysis can be carried on. Apparently, this implies that a significant effort is required to the testing engineer for the definition of the testing inputs. On the contrary, one of the main strengths of ST is to include a coverage criteria that requires minimal assumptions on the testing inputs definition without loss of guarantees.

Exhaustively listing all the system inputs in an adaptive system can be a difficult task [34]. In some cases, the test input definition problem can be circumvented by executing the actual system with the actual inputs, rather using synthetic random inputs. In the web application example, one could collect traces from the execution of the actual software and analyse this data as a set of random test cases. If the executions are collected systematically (i.e., two different system executions will differ exclusively by the unknown inputs), the uncertain inputs are in this way *by definition* representative of a possible realization of the adaptive system. This is a viable solution thanks to the minimal assumptions on the distribution of the inputs required by ST.For this reason, in this paper we build on an approach that does not require any of such assumptions. As we discuss formally in Section 4, the fact of not requiring any assumption on the input probability distributions is one of the strengths of the proposed scenario theory.

Conversely, in our testing set-up, all the inputs that are

1. The implementation of the experiments presented in the paper is publicly available and has been reproduced through the conference artifact review process https://github.com/ManCla/ESEC-FSE-2020 [33].

known and fixed once the system is implemented should not be randomised. In the web application example, this could be the number of servers on which the applications is deployed. If the number is known and fixed at deployment-time, their number should be fixed also in the tests. Otherwise, if the application is expected to adapt to a varying number of severs, their number should be randomised during and across the tests.

In order to evaluate the effectiveness of the adaptation strategy, we define a *performance parameter* that we compute for each test. The performance parameter is a quantity that (i) can be measured from the execution of a test case, and (ii) is higher or lower, according to the degree at which the adaptation strategy has achieved its goals. In the web-application case mentioned above, this parameter could be, for example, the average time spent recovering from server failures over the whole test duration. The key intuition is that this performance parameter is itself a random variable [25], and we can therefore use tools from statistics to deduce properties of its value.

The extraction of such properties can be done in different ways. Traditional statistics offers different tools that could be considered. We argue that these tools present fundamental limitations that hinder their applicability to test self-adaptive software. As an alternative, we propose the use of scenario theory.

Hence, we collect the outcomes of the tests and evaluate them using the scenario theory. By leveraging this theory we obtain *probabilistic bounds* on the chosen performance metric and a *testing confidence*. The probabilistic bounds are in the form of a minimum performance that is guaranteed in a high percentage of the cases. The testing confidence is given as a probability. To be precise, the confidence is the probability that we have missed relevant test cases that would have changed the obtained bound. Continuing with the example above, we would obtain a bound like "*the time it takes to recover from a server failure is on average less than* 42 *seconds in* 97% *of the cases, with a* 95% *confidence*". This means that we have a $100 - 95 = 5\%$ probability of having missed a relevant test case. If the confidence is not sufficient, the scenario theory allows the testing engineer to directly compute how many additional tests are needed to increase it to the desired level.

3 Background and Related work

This section discusses how this work is connected to the existing research literature. To start, we present related work in the software testing research field. Then we present the traditional statistical tools used to extract probabilistic properties from test outcomes.

3.1 Testing of Adaptive Systems

Our work connects to different areas of the existing software testing literature: (i) testing of self-adaptive and context-aware systems, (ii) testing in the presence of environmental dependencies, (iii) fuzz testing, and (iv) testing for probabilistic guarantees.

The problem of testing an adaptive software – in some cases also called *context-aware* software [22], [35] – is not a new challenge for the software testing community [14], [16]. We split the work that addresses the testing of self adaptive software in *design-time* and *run-time* approaches. For self-adaptive software, the design-time approaches include SIT [36] and TestDAS [37]. SIT [36] proposes a test case generation technique for self-adaptive applications. The sampling of the input space is based on an interactive model of the application that is being tested. TestDAS [37] focuses on triggering the adaptations during the test cases. It leverages models of the software behaviour that are defined in advance by the programmer. Context-aware software is close to self-adaptive software, and there is a significant amount of work addressing the problem of testing context-aware applications [22], [35], [38], [39]. The self-adaptive (or context-aware) software observes the execution environment and selects actions to be performed based on the result of the observation phase. The research effort for context-aware software goes in the direction of generating test cases that trigger the context-aware software layer [22], [38], [39]. In [38], automatically generated bigraphs are used to model the interactions between the environment and the software, and to generate the test cases. In [39] the authors propose a framework for automatically generating test cases with high-level test data.

Our proposal is different from previous work on context-aware and self-adaptive software testing, since in our case the interaction with the environment only needs a probabilistic characterisation, and no further modelling effort. Moreover, in our contribution, the number of test cases does not depend on how the interaction with the environment is performed. This is important since it allows our method to scale with the amount of interaction between software and environment.

The literature on software testing also includes efforts to develop run-time testing methodologies for adaptive software [3], [40], [41]. Generally speaking, there is a need to develop models for verification and validation at run-time [3]. This need is caused by the ever-changing nature of the environment the adaptive software operates in. We describe our approach for design-time testing, but in principle[2] the resulting method can be applied during the run-time execution of the software application, since it only requires data collection and analysis. A clear difference between our work and the related literature is that we develop a probabilistic approach.

In our work, we use statistical tools to evaluate the performance of the adaptation layer of a self-adaptive software, independently from changes in the environment. Previous work also addressed the problem of testing a software regardless of its environmental dependencies [42], [43]. These works aim at decoupling the tests outcomes from such dependencies. To test the adaptation layer, we need to preserve the dependency on the environment, since it triggers the need for adaptation. However, we aim at obtaining an evaluation that is general with respect to the environment changes.

2. The requirement to apply our approach at run-time is that the run-time tests are considered random independent tests. Testing the system continuously might not guarantee independence. This can be solved (for example) by introducing a delay between consecutive tests.

The approach we propose in this paper is based on random sampling of the system inputs and environment scenarios. This practice is known to the software testing community [44], [45], and is often called fuzz testing [25], [46], [47], [48]. The literature focuses on using random generation for achieving adequate exploration of the software behaviour, e.g., code coverage [48]. We take inspiration from fuzz testing, and use random sampling with two different objectives: (i) decoupling given inputs or environmental scenarios from performance parameters that indicate how well the adaptation layer is performing, and (ii) obtaining a probabilistic characterisation of the performance metric.

Probabilistic guarantees have been explored [49], [50], [51]. In some cases this exploration targeted approximate computing [17], [26], [52], [53], which is not the subject of this study. Some existing work target service-oriented software architectures [54] and how to combine the probabilistic guarantees given by the different services to obtain guarantees for the complete system [50], [51]. Recent work used a probabilistic approach to compensate for the uncertainty of the dependence between system configurations and system performance [55]. However, no prior work targets dynamic behaviour (i.e., behaviour that changes during the execution of the software, as it is the case with the adaptation layer) and adaptive software, which is the focus of our work.

3.2 Tools from Statistics

In this section, we recall traditional tools from statistics, that could be used to analyse the result of tests and provide statistical guarantees on the software behaviour. We both describe the underlying theory and their application. A throughout discussion of the limitations that make them unsuitable for the testing of self-adaptive software is later presented in Section 4.1. We choose Monte Carlo Sampling (MC) and Extreme Value Theory (EVT) due to their application in software testing, simulations, and rare event analysis. Statistics offer additional tools, e.g., martingale and black swan theory, that are less suited to the analysis of a vast corpus of data or require additional knowledge. We have not found evidence of application of other theoretical results to the field of software testing.

Monte Carlo Sampling (MC): Monte Carlo (MC) methods [27] use repeated random sampling and simulation to numerically predict the value of parameters. The parameters are unknown, and usually no exact analysis can be carried out (for example because there are too many random variables, i.e., too much uncertainty). Nowadays, MC methods are employed in many different fields, from optimisation [56] to decision making [57]. MC methods leverage the Central Limit Theorem [58] as a main mathematical result. The theorem discusses the mean of a random variable with an arbitrary probability distribution, under the assumption that the variance of the distribution is finite. The theorem states that, if one draws infinitely many samples from the random variable, the distribution of the arithmetic mean of the samples asymptotically converges to a normal distribution, regardless of the original variable distribution.

The application of MC approaches allows to conduct an arbitrary number n of tests and measure the random variable X, obtaining a set of outcomes $\{x_1, \dots, x_n\}$. Then it is possible to determine the mean value $\bar{x}$ as the arithmetic average of the tests outputs,

$$\bar{x} = {}^{1}\!/_{n} \sum\nolimits_{i=1}^{n} x_i. \tag{1}$$

The computed arithmetic mean $\bar{x}$ is also a random variable. The Central Limit Theorem guarantees that its distribution converges to a normal distribution for increasing n, i.e., $\bar{x} \sim \mathcal{N}(E[X], {}^{\sigma^2}\!/_{n})$, where $E[X]$ is the expected value of the random variable X and σ^2 is the variance of X. When n is big enough, the observed mean value converges to the actual expected value for the quantity of interest. This result is well-known in statistics and it holds irrespective of the specific software application under test. In fact, convergence is guaranteed independently from the probability distribution of the performance metric. However, there is no general result on the speed of the convergence and it is therefore application-dependant. With MC sampling, the significance of the test results and the choice of n is therefore left as an arbitrary choice to the testing engineer.

MC methods are therefore naturally used to evaluate the *average behaviour* of a system. By average behaviour $\bar{x}$ we mean a performance that best summarizes the different possible performances that the system can expose.[3] The standard deviation σ complements this information by quantifying instead the *spread* of the possible performance. By spread we mean the width of the range of possible performance values. The standard deviation can be estimated using the *sampled standard deviation*:

$$\bar{\sigma}^2 = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{n-1}}. \tag{2}$$

Intuitively, since the sampled mean is a description of the average behaviour, each randomly generated test carries significant information about it. For this reason relatively few test can provide already a good convergence of the sampled mean $\bar{x}$ to the expected value $E[x]$. This is still application dependant and there is no general approach for quantifying this convergence.

Using MC methods we could be able to state, using again the recovery time example from Section 2, that *if the variance is sufficiently low, the system in is most likely recovering from a server failure in a time close to* 42 *seconds*. Unfortunately the theory doesn't allow us to rigorously quantify the "*most likely*" words used in the statement. In the same way, also the word "*close*" cannot be generally quantified rigorously. Finally, we cannot quantify the confidence that we can have in this statement. In fact, the confidence depends on whether the number of tests is sufficient to apply the central limit theorem or not.

MC methods have found limited use in the context of software testing [59], [60]. None of these works focuses on the testing of self-adaptive software. In [60] MC methods are used to test the reliability of a software system, while [59]

3. Formally, the average corresponds to a normalized sum of the possible outcomes weighted by their probability. In general, the average value is not necessarily the most *probable* outcome, and it is not even guaranteed to be an actually *possible* outcome. For example, if we average the number of attempts necessary to obtain a response from a server, the average will likely have decimal values, but the possible outcome is only an integer. These aspects also limit the effectiveness of analyzing a performance metric by looking at its (sampled) average.

generally discusses how MC methods can be applied to software testing.

Extreme Value Theory (EVT): The Extreme Value Theory [61] (EVT) study a random variable around the tails of its distribution. This is opposed to MC methods that study the behaviour of a variable around its average. EVT could therefore be used when we specifically want to analyse the software's *worst-case* behaviour, e.g., what is the maximum memory occupation of a program. The theory is nowadays widely adopted to study rare phenomena such as earthquakes, quantitative risks in finance, but also extreme events in engineering [62], [63].

The role of the Central Limit Theorem for MC sampling is taken by the Fisher–Tippett–Gnedenko Theorem [64] for the EVT. The Fisher–Tippett–Gnedenko theorem defines the family of distributions to which the maximum value of a set of samples converges. The family of distributions is called the Generalised Extreme Value Distribution (GEVD) [61]. To apply EVT, we can look at a set of data (in our case the performance parameters obtained from the test cases) and extract a set of samples that belong to the tail of the dataset – i.e., a set of *maxima*. We then fit the the GEVD to the extracted maxima. In this way, we can obtain a probability distribution for the extreme value of the performance metric that could be observed in future executions of the system.

There are different practices to extract the maxima from a dataset. The most common are: (i) the Block Maxima, and (ii) the Peaks Over Threshold. The former defines a partition of the dataset and extracts the maximum value from each subset, the latter takes all the values that exceed some predefined threshold. The difference between the two methods stems from the possibility of partitioning the dataset or not (for example in smaller sets of data – acquired with different software releases). When the data is naturally partitioned into smaller sets, the block maxima methods is preferred. Since in our case each data belongs to the same partition we use the peak over threshold method.

Using the obtained maxima we can estimate the parameters of the GEVD. This distribution has the following form:

$$f_{GEVD}(x) = \frac{1}{\sigma}\left[1 + \left(\frac{\xi\,(x-\mu)}{\sigma}\right)^{-\,(\xi\,+\,1)/\xi}\right], \tag{3}$$

where μ is the location parameter, σ is the scale parameter and ξ is the shape parameter. The location parameter defines the starting point, the scale parameter defines the weight of the tail and the shape parameter defines the rate of decrease of the probability density.

With the obtained distribution we can evaluate a probabilistic bounds τ on the performance of the system, i.e. $P(x > \beta)$. To do so we need to account for the probability that a value is actually a maximum and the probability of that maximum being greater than the bound. This can be computed multiplying the two probabilities:

$$P(x > \beta) = P(x > \beta | x > \tau) \cdot P(x > \tau), \tag{4}$$

where τ is the threshold used for the selection of the maxima. The first term in the multiplication is equivalent to the cumulative probability distribution of the GEVD. The latter can be estimated as the ratio between the total number of performed tests and the number of test that resulted in one of the maxima:

$$P(x > \tau) \approx n_{maxima}/n_{tests}.$$

In this way, we can use EVT to obtain a probabilistic bound for the performance of the system under test. When applied to the example from Section 2, we would be able to provide guarantees in the form: *there is a* 1.5% *probability that the recovery time is worse than 50 seconds*. If needed, a different numerical bound can be given – apparently with a different associated probability. This is due to the fact that EVT allows us to retrieve a complete distribution for the maximum of the performance.

EVT presents similar limitations compared to MC approaches [65]. EVT uses an arbitrary number of samples from the distribution of interest. Moreover, the choice of which and how many samples can be considered as the maxima is also arbitrary. There are also no results on the rate of convergence of the samples to the Generalised Extreme Value Distribution. Conversely to the MC methods that talk about the average case EVT discusses the rare cases. For this reason few tests carry information about such cases. Therefore the convergence to the GEVD is generally slow. Finally, as MC, EVT requires finite variance of the parameter that is sampled.

4 Methodology

In this section, we describe our approach to obtain probabilistic guarantees and its theoretical underpinning.

4.1 Limitations of Traditional Statistics

In Section 3.2 we described the traditional tools from statistics that could be used to obtain probabilistic guarantees when testing self-adaptive software: MC and EVT. Both those methodologies suffer from limitations that make them inconvenient for analysing the results of the testing campaign – i.e. being used in place of the "Scenario theory" block in Figure 1. These limitations are: (i) arbitrary choice of testing parameters, (ii) unknown, case-dependent, testing confidence (or *testing adequacy*), and (iii) assumption that the variance of the measured quantity is finite.

MC and EVT use an arbitrary number of samples for the desired estimation. The MC approach assumes that the set of samples is large enough that the Central Limit Theorem holds [27], and the EVT similarly relies on the convergence of the maxima samples to the Extreme Value Distribution [66]. Unfortunately, in both the theories, there is no general way to define how many samples are needed to achieve convergence.

The impossibility of quantifying the convergence to the Gaussian and Extreme Value Distributions has another relevant implication for the testing problem. If the desired testing confidence is not reached, it is impossible to quantify how many extra tests are needed to reach it. In other words, we cannot know *a priori* how the confidence will change when performing one extra test.

Another assumption needed by both EVT and MC sampling is that the performance parameter has finite variance. In practice, this means that either the probability of it being infinite must be very low, or that the parameter can only take finite values. Suppose we are trying to assess the worst-case execution time of a software function. The presence of a bug could cause the processor to stall and the function to never terminate. As long as the occurrence of this bug is sporadic, it is possible to use EVT and MC to determine metrics on the execution time. However, if the bug is triggered more often, the (higher) probability of an infinite execution time would prevent us from applying EVT and MC methods. Some commonly used engineering solutions can enforce finite variance in given performance parameters. An example of this is the presence of timeouts. Introducing a timeout does not help overcoming the limitation. In fact, the test that resulted in a timeout does not provide a sample of the possible performance of the system (i.e., conveys less information than its number-based counterpart, resulting only in a 'timeout reached' outcome). Merging this information in the statistical evaluation is non-trivial, and could even be detrimental and hide behaviours of the system.

Our proposal overcomes these limitations by formulating the testing problem as an infinite optimisation problem and solving it using the *scenario theory*.

4.2 Scenario Theory for Software Testing

The *scenario approach* [29] was developed in the field of robust control [67]. However, it is more generally applicable than control design. It provides a method to solve *infinite convex optimisation problems*. Infinite convex optimisation problems are a class of optimisation problems that appear frequently in robust control design. However, they are also classically found in other fields, such as decision making, finance, and management [68], [69]. The contribution of this paper is the formulation of the testing problem with the scenario approach and the study of the results that can be obtained for self-adaptive software. We show how this allows us to overcome the research challenges from Section 1.

In our testing problem, we want to find bounds for a performance parameter of an adaptive system (i.e., of the software and a given adaptation strategy implemented on top of it). In general, finding a safe and very pessimistic bound on what the software can achieve is trivial. The interesting question is how much we can move this bound toward higher performance. This problem can be formulated as: we would like to maximise the value of the performance parameter that we can safely guarantee when using a given adaptation algorithm.

The evaluation of this performance bound can therefore be seen as an optimisation problem. Solving optimisation problems means finding the extreme value of a quantity, either the highest or the lowest possible. In the following sections we introduce optimisation problems, the scenario theory, and how they can be used to bound the performance of a self-adaptive software.

Optimisation Problems: Optimisation problems are defined by: (i) one or more decision variables, (ii) a cost function, and (iii) a set of constraints. The decision variables are the quantities we can choose and change. The cost function is the quantity we would like to maximise or minimise, and it should be a function of the decision variables. The constraints are statements about the decision variables that we want our final solution to satisfy. An example of a problem that can be formulated as an optimisation problem is the travelling salesman problem [70]. A salesman needs to determine a route to visit a given number of cities, minimising the travelling distance. The decision variables are the segments to add to the path, the cost function is the total travelled distance, and the constraint is that all the cities in the given list are visited at least once.

In our proposed testing methodology the decision variable is the worst-case performance of the adaptation strategy (i.e. the best value of the performance metric that we can safely guarantee), the cost function is the worst-case performance itself, and each of the test outcomes is a constraint. The performance bound evaluation therefore becomes the following optimisation problem: *maximise the performance that can always be guaranteed, under the constraint that it cannot exceed what is experienced in the conducted tests*.

Being even more practical and using the web application example from Section 1, suppose we want to provide guarantees on its maximum response time thanks to the adaptation strategy. We conduct a certain number n of tests. Each test is composed of servicing 1000 requests in random execution instances of the overall system, and monitoring their response times. We record the average response times in the vector $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$. Where r_i is the average response time of the web application for the 1000 requests of the $i-th$ test. These values are constraints on what the software can achieve. We then take the maximum element of the vector as our worst-case performance metric, $w_{\max} = \max\{r_1, r_2, \ldots, r_n\}$. If we tested all the possible execution cases, we could then say that we guarantee that the response time will be lower than the maximum value $w_{\max}$. However, for self-adaptive software the set of possible execution cases is likely infinite.

Ideally, if we could perform an infinite number of tests, we would test the system in every possible situation. In this way, we could obtain an exact evaluation of the worst case behaviour of the system. In practice, this is apparently not achievable, and we have to rely on only a finite number of tests. Despite this, when the number of tests is sufficiently large, it will still provide significant information about the general case.

Infinite Optimisation Problems: If we cast our (ideal) testing problem into an optimisation problem, we would have infinite constraints (the infinite test cases). For our web application example this would mean performing an infinite number of tests and obtaining the real bound. Unfortunately, solving an optimisation problem with an infinite set of constraints is not always possible (or desirable). Similarly, in our testing problem, we cannot perform infinite tests.

Scenario theory: The scenario theory [29] addresses the problem of solving an infinite optimisation problem while accounting only for a finite number of the constraints. The theory provides probabilistic guarantees on the generality of the solution. The scenario approach is used to solve infinite optimisation problems. The approach is to transform the infinite-sized problem into a finite-sized problem by *randomly* sampling a finite number of constraints from the infinite set of possible ones. Then, it is possible to solve the optimisation problem accounting only for the finite set of sampled constraints. The scenario theory allows then to quantify the uncertainty and the guarantees that are lost by only considering the finite set.

In the testing of our web application, this corresponds to obtaining the probabilistic guarantee that the average response time is lower or equal to $w_{\max}$ in a high percentage of the cases. This means that, with high probability, the future executions of the web application would not result in a higher average response time, i.e., $\mathrm{P}(r_m \leq w_{\max} \mid m > n) = p_w \approx 1$.

Using the scenario theory, we can compute the probability p_w that the solution – computed using the finite set – does not satisfy all the constraints in the possibly infinite set. For our testing problem, this means that we evaluate the worst-case performance using only a finite number n of test results. We then compute the probability that the obtained worst-case value $w_{\max}$ holds for all of the infinite tests that we could possibly run – i.e., we compute the probability that in future tests we would obtain a worse value than $w_{\max}$, which is obtained using the first n tests, i.e., that $\exists m > n \mid w_m > w_{\max}$.

In our specific optimisation problem for testing, we have only one decision variable (the evaluation of our worst-case). We now state the relevant result of the scenario theory in that case.[4] We denote with ε the probability of *observing (in future executions) a performance value that is worse than the observed worst-case up to* n *tests* (i.e., $\varepsilon = 1 - p_w$). In the original optimisation framework this is the probability of not satisfying all the infinite constraints.

Using the scenario theory, we can evaluate the probability that, in our n test cases, we could have *missed a test case with a worse performance than the obtained bound*. We call this probability β and it is computed from ε and n as

$$(1 - \varepsilon)^n = \beta. \tag{5}$$

In the original optimisation problem, ε quantifies the probability that a new (randomly picked) constraint taken from the infinite set would invalidate the solution found using the finite set. In our testing analogy, ε is a quantification of how *tight* we want our bound to be. Choosing a lower probability ε means having a tighter bound, and choosing a higher value means that we allow for higher risk of not having observed the *true* worst-case. We remark that we can arbitrarily choose ε, but this will result in different degrees of confidence β that we can have in the obtained result.

The probability β can be seen as a quantification of how confident we are of our testing campaign result. A lower value of β implies that we are more confident and a higher value represents a higher probability that the final result is not correct. A tighter worst-case bound (lower ε), in fact, results in a higher β, a higher probability that we could have "missed" a relevant test case (constraint) in our sampling. In this sense, β can be seen as a *coverage* parameter, since it quantifies our confidence of having explored enough of the possible instances of the self-adaptive software behaviour.

Figure 2 shows a graphical interpretation of the probabilities ε and β. The dashed line shows the true and unknown probability distribution of the performance parameter. The histogram represents the observations that we obtained when measuring the performance of the system in our tests (i.e., our test results). The red bar indicates the worst case obtained during the testing campaign. The red area has size ε, i.e., ε is the probability that in the future we will experience a worst performance than the observed worst-case. Here, β is the probability that – assuming that ε is the correct area – we would not have observed a test case in the ε area during our n observations. For example, if we had more test results, these could or could not be lower than the observed worst-case. In any case, with more observations, we are able to: (i) tighten the bound (i.e., decrease ε), (ii) increase the confidence (i.e., decrease β), or (iii) do both things to a lesser extent. Without running additional tests, we can tighten the bound at the cost of losing confidence in it. Alternatively, we could loosen the bound and increase our confidence.

4. We omit the complete formula for an arbitrary number of decision variables, since it is not of interest in our case.

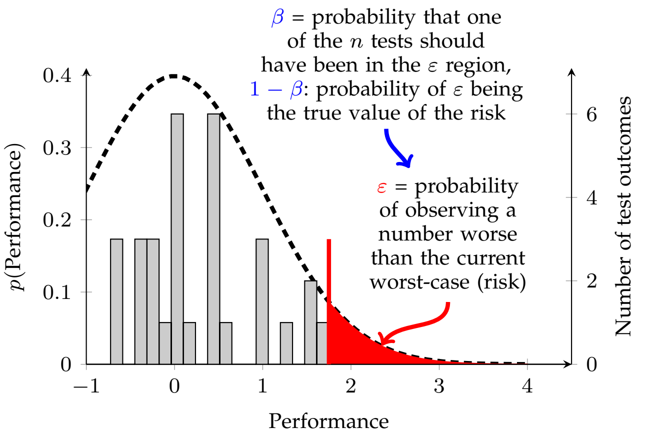

Fig. 2. Graphical representation of the scenario parameters ε and β. In this case poor performance of the system is captured by high values of the performance parameter while low values correspond to more desirable performance. The histogram bars represent the performance observed in the test cases. The red bar is the observed word case. The red area corresponds to the probability ε quantified with ST of observing (in future executions) a performance worse than the worse performance observed so far. Figure from [24].

We highlight that the theory does *not* require any prior knowledge on the probability distribution of the performance metric (i.e., on the dashed line in Figure 2). This is the strength of scenario theory with respect to the traditional methods that require assumptions on this probability distribution (e.g., its variance being finite).

We also remark that the test cases have to be randomly generated (or taken from the execution of the software in different scenarios). This is what guarantees the probabilistic characterisation. It could be argued, in fact, that what is actually used is only the test case where the system exposed the worst behaviour, and therefore this one is the only test case of interest. But identifying the testing conditions that expose the worst case might be not be straightforward and could require a greater effort than running a number of randomly generated test cases. In other cases, instead, the worst-case performance could be a trivial, arbitrarily bad performance. For example, the worst case response time of a web service will be infinite if all the servers become unavailable. Differently, we ask instead the following question:

given a number of tests we ran on the real system, what is the average response time that we can guarantee in 99% of the cases? We argue that this probabilistic characterisation of self-adaptive software is (i) simpler to achieve and, (ii) more interesting than its deterministic counterpart. Therefore, when taking the probabilistic approach, even though a new test might not change the worst-case bound, it is still valuable because it increases the reliability and confidence in the obtained bound.

This probabilistic characterisation of the guarantees specifically addresses the research challenge **CH1**. Our argument is that, since deterministic guarantees cannot be given for adaptive systems, we should aim for probabilistic ones. Within the choice of probabilistic guarantees we have then addressed the other two research challenges. In fact, we have showed how to apply scenario theory for quantifying the system performance and testing confidence. Respectively, ε quantifies the probabilistic bound on the performance (**CH2**), and β quantifies the testing adequacy (**CH3**).

5 Experiments

This section aims at validating the proposed methodology. Our approach is designed to: (i) provide formal probabilistic guarantees from experiments (**CH1**), (ii) allow us to perform a fair comparison of different adaptation strategies (**CH2**), (iii) quantify the trade-off between the number (and cost) of experiments and the obtained probabilistic confidence (**CH3**). Finally, every method has to take into account the choice of test inputs. Hence, we explain how the choice of randomized testing inputs can affect the results of the testing campaign (**CH4**)

The proposed approach (shown in Figure 1) is *application independent*. We highlight this strength presenting experimental data from well-established adaptive software with different application domains: healthcare, video processing, and traffic flow optimisation.

In this section we show two different applications of the approach using the three presented analysis tools: MC, EVT, and ST. We highlight and discuss the respective limitations and what each technique can be used for.

In particular, in Section 5.1 we focus the discussion on the trade-off between the number of performed tests and the obtained probabilistic confidence using a simulation tool for the Tele Assistance Service (TAS) [30], [71]. This shows how we address the research challenges **CH1** and **CH3**. In Section 5.2 we focus on the comparison of different adaptation strategies using the Self-Adaptive Video Encoder (SAVE) [31]. Our approach allows us to address the research challenge **CH2**. Furthermore we include a discussion on the possible consequences of a partition of the test inputs, this concerns the discussion the research challenge **CH4**. In Section 5.3 we use the TRAPP case study to discuss the role of test inputs definition in our testing methodology: this discussion addresses the research challenge **CH4**.

5.1 Data vs. Confidence Trade-Off

Aim: The aim of this case-study is to discuss the different probabilistic guarantees that can be provided with the different methodologies. To do so we show the complete application of the different methodologies (MC, EVT and ST) to the TAS system and comment the results. The results expose, among other facts, the limitations of the traditional statistical approaches in terms of confidence evaluation. Conversely, we discuss the probabilistic guarantees obtained with ST: the probabilistic performance bound and the testing confidence are discussed. More specifically, when using ST, we can offer guarantees on the software performance level, even for test cases that have not been explicitly executed (**CH1**), and we evidence the direct connection between the amount of collected experimental data and the probabilistic testing confidence (**CH3**).

Self-Adaptive Software: TAS is a service-oriented software application that provides care and assistance to elderly people that suffer from chronic diseases [71]. The software [30] periodically monitors patients conditions using sensors and activates a chain of services invocations. First, the patient conditions are sent to an *Analysis Service*, that inspects the data and determines the next steps to be taken for the patient well-being. The outcome of the analysis is one of the following: (i) do nothing, (ii) invoke a *Drug Service* that will compute a new medicine dosage, or (iii) invoke an *Alarm Service* that will dispatch an ambulance.

Each service can be realised by multiple service providers, potentially doing different computations that follow the same specification and interface. During the execution of the software, the selection of which provider to invoke to obtain a given functionality introduces an element of choice in the management of each request. Service providers have different properties; e.g., quality of the service, availability, success rate, and failure probability. In our experiments we focus on service rate and availability in the presence of failures, i.e., the number of requests processed per time unit and the probability of serving incoming request successfully.

The presence of different service providers and variety of potential needs for each request introduces the need to adapt the software behaviour to the current operating conditions. Adaptation strategies were introduced with the aim of selecting given services based on properties to be enforced for the overall system, e.g., [9], [72], [73], [74]. In our experiments, the adaptation strategy should recognise the service providers with higher service rates and prioritise them when distributing the requests. Also, since services might not always be available, the adaptation layer should avoid submitting requests to unavailable service providers.

To identify the best choices, the adaptation layer stores one number per service provider, called *weight*. For all the alternatives, the weight is initialised to 1 and incremented or decremented (using a fixed step equal to 50 in our experiments) based on the service performance. For each successfully processed request, the weight increases, and for each failed invocation the weight decreases. We further introduce a timeout and reset the weight to 1 if the service invocation failed for all the requests sent in the timeout interval.

When distributing the requests, the weights are used to define a probability distribution over the different providers of a given service. The probability distribution can be obtained by normalising each of the weights over their overall sum. The requests are distributed according to this

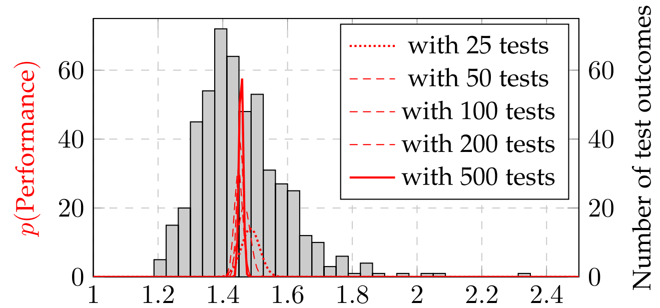

Fig. 3. TAS: Histogram of experienced average number of attempts and plot of the obtained normal distribution of the sampled average performance.

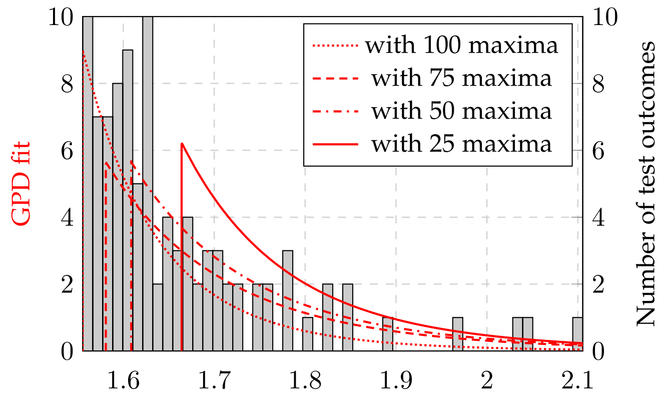

Fig. 4. TAS: Histogram of all the test outcomes that exceed 1.5554 average attempts per request. In red are plotted the GEVDs fitted for different maxima choices in the TAS case study – namely 100, 75, 50, 25 maxima.

probability distribution. We limit the weights to an interval between 1 and 1000. This avoids that overly positive weights attract all the requests. In the same way, negative weights imply that the service provider is never chosen, making it impossible to recover even in case of potentially correct operation.

Test Design: We use the TAS case study to highlight the trade off between data and confidence, i.e., how the exploration of the system's behaviour improves with the increasing number of tests. The definition of which inputs should be randomised is critical for the correct coverage of the system's behaviour. Here, we randomise: (i) the requests profile, i.e., the number of incoming requests; (ii) the workload mix, i.e., the type of incoming requests; (iii) the availability of the different service providers, i.e., a provider being reachable or not; and (iv) the reliability of the service providers, i.e., request processing may fail due to internal reasons.

We can use one or more performance parameters, depending on the specific software and on what are the aspects that we want to test. The performance parameter should be representative of the behaviour of the adaptation layer. Practically, this means that it should enable the distinction of whether the adaptation layer worked well for the specific test case, or not. In the TAS case we want to build a system that is robust to the occurrence of failures. We choose as performance parameter the average number of attempts needed for a request to be correctly handled. Lower numbers indicate better adaptation, 1 being the best possible value (often not achievable).

Results MC: Figure 3 shows the histogram of the obtained worst case in the different tests. For each possible performance value on the horizontal axis the column above is proportional to the number of tests that exposed that performance. In the same figure, we also plot the gaussian distributions obtained for the sampled mean and its sampled variance using an increasing numbers of test cases.

We use this figure to investigate what MC methods allow us to state about the average performance of the system and comment on its applicability. The obtained gaussian distributions do not change significantly when increasing the number of tests – i.e., the different line plots are close to each other. This shows the quick convergence of the MC methods: already with 25 tests *we obtain some confidence that we can expect (on average) a performance of around* 1.45 *number of attempts with this adaptation strategy*.

On the other side, this evaluation does not give formal guarantees on such statement. Specifically, we would like to quantify the words "*around*" and "*some confidence*" from the statement above, but with MC this cannot be done in the general case. As an example, we cannot state that 1.45 is the performance that we are most likely to observe. The most probable performance, in fact, seems to be slightly lower according to the histogram.

To summarise, the experiments show that MC methods can be used to get a rough evaluation of the average behaviour of an adaptive system. Their quick convergence allows to achieve this with relatively few experiments. But, if the purpose of the testing campaign is to formally constrain the adaptation performance, they lack of a general approach to the evaluation of the testing confidence.

Results EVT: Figure 4 shows the histogram of 100 maxima from the testset of the TAS system. Overlying to the histogram, are plotted the GEVDs fitted using different numbers of maxima, namely: 100, 75, 50, and 25. The application of EVT provides a full probabilistic characterization of what can possibly be the worst case performance of the system.

If we take one set of maxima and the associated fitted distribution we can provide probabilistic guarantees on what is the worst case performance of the adaptation strategy. For example if we take 1.9 as candidate worst case the fitted distributions can be used to compute the probability of obaining a performance worse than that. Using the distribution fitted to 100 maxima we would obtain *a* 0.92% *probability of obtaining a performance worse than* 1.9. The same statement could be made for different candidates for the worst case performance (apparently associated to a different probability).

Unfortunately, this evaluation doesn't include a quantification of the confidence we can have on the statement. This depends on whether the used maxima are sufficient to obtain convergence to the GEVD and the theory doesn't provide a way to quantify it. Experimental evidence of such statement is that the different choices of the maxima provide different results for the worst case probability. Using 75 maxima we would have in fact obtained a probability of 0.73%, using 50 maxima we would have obtained a proba-

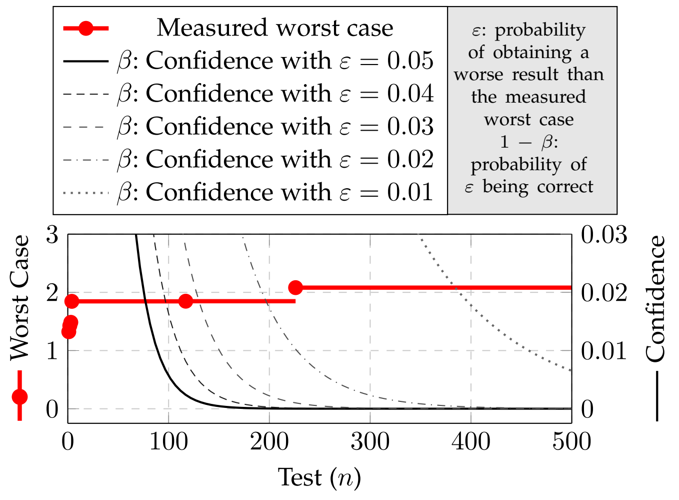

Fig. 5. TAS: Measured worst case and confidence level varying the number of performed tests using the Scenario Theory. Figure from [24].

bility of 0.58%, or using 25 maxima we would have in fact obtained a probability of 0.37%. The theory doesn't provide a way to evaluate the confidence and therefore to state which of those choices can be considered more appropriate or reliable.

Results ST: Figure 5 shows the evolution of our quantities of interest when we perform an increasing number of tests and analyse them using ST. In particular, it shows: (i) the worst experienced average number of attempts needed per request (using the left y-axis), and (ii) the confidence β in the test outcome for different values of ε (using the right y-axis).

In the figure, we highlight with circle markers the newly experienced worst cases. The worst case is monotonically increasing with the number of conducted experiments. For example, in test #226, the average number of attempts per request to complete the TAS cycle is 2.081. This is a new worst case, as the previously experienced value was 1.8479 (from test #117).

The probability of not performing a relevant test (i.e., a test that would lead to a different worst case) is monotonically decreasing with the number of performed experiments. Analogously, a higher number of test cases is leading to a higher test coverage. Despite an unchanged worst case, between tests #117 and test #226, our confidence in the experimental results grew (lower values of β).

Decreasing the value of ε means being more conservative with our evaluation. The non-solid lines show the confidence β with smaller values of ε (up to 1%). Many more experiments are needed to obtain the same level of confidence when a smaller ε is selected.

The quantity β is the key difference between ST, and EVT or MC. Within ST, the test brings information both on the reliability of the testing process and the system under test itself. In MC and EVT the information carried by the tests is used only to evaluate the system's performance. Differently from MC and EVT the testing confidence allows the testing engineer to make a conscious choice on the number of randomly generated test cases according to the needed testing confidence and performance evaluation.

Using the scenario theory, we can state:

> *Based on the results of $n = 500$ tests, requests sent to TAS (with the described adaptation strategy) will not need more than 2.081 attempts on average to complete (despite service failures) with probability $1 - \varepsilon = 0.98$. This statement is correct with probability $1 - \beta = 0.99996$.*

This performance is apparently strongly dependant on the chosen adaptation strategy. More interestingly, it does not depend on the specific values of the quantities that have been randomised for the test case generation. Conversely, we could determine the number of tests to be performed based on the desired ε and β values:

> *Given the desired probabilistic guarantees of confidence of $1 - \beta = 0.99996$ and a bound that holds in 98% of the cases, we perform $n = 500$ tests. In our case, the 500 tests indicate that in the worst case 2.081 attempts are needed on average per request.*

Suppose that we could afford to conduct only $n = 250$ tests. In Figure 5 we can see that the measured worst case is the same as the complete test campaign. However, keeping $1 - \varepsilon = 0.98$, we could only claim a lower confidence in our test findings:

> *Based on the results of $n = 250$ tests, requests sent to TAS (with the described adaptation strategy) will not need more than 2.081 attempts on average to complete (despite service failures) with probability $1 - \varepsilon = 0.98$. This statement is correct with probability $1 - \beta = 0.9936$.*

Vice versa, we could also determine the larger bound ε that we need to accept for if we wanted the same confidence $1 - \beta = 0.99996$ for 250 experiments. In this case we would obtain $1 - \varepsilon = 0.9603$.

5.2 Adaptation Strategies Comparison

Aim: The aim of this second set of experiments is to show the use of the proposed methodology for the comparison of different adaptation strategies. We run the tests and quantify the performance for each case with all the three discussed tools. The experiments expose the limitations of MC and EVT in enabling fair comparison. This is achieved, instead, with the use of ST (**CH2**). We aim at using the presented statistical tools to compare in a fair way the different adaptation strategies. Moreover, we also show the application of the scenario theory for testing with different and conflicting adaptation requirements (**CH1**). To further emphasise the validity of the proposed methodology, in this section we run the tests using the real software, rather than a simulation tool.

Self-Adaptive Software: SAVE [31] is a video encoding tool that aims at automatically achieving the desired size compression of a video stream whilst preserving as much as possible of its content. We target video broadcasting services, where multiple videos are streamed with a fixed amount of bandwidth and unpredictable demands. We also assume that the video content is not known a priori and is expected to change over time. The need for adaptation arises from the strong dependence of the encoding performance on the specific content of the video.

The adaptation strategy should leverage the frame characteristics to autonomously find an effective combination of encoding parameters. For each frame, the adaptation layer selects: (i) the *quality* parameter that specifies the compression density. It ranges between 1 and 100, where 100 preserves all frame details and 1 produces the highest compression; (ii) the *sharpen* parameter, which specifies the size of a sharpening filter to be applied to the image. The filter size ranges between 0 and 5 where 0 indicates no sharpening; (iii) *noise correction*, which specifies the size of a noise reduction filter, also between 0 and 5. High filtering should in general generate a more uniform image, making it simpler to compress.

For each frame the adaptation layer measures size and quality and selects the encoding parameters accordingly, using its own algorithm. The size is measured in bytes and the quality is measured using the Structural Similarity (SSIM) index [75]. This index is a unitless metric that ranges between 0 and 1 and quantifies the similarity between the original and the encoded frame (high index meaning high similarity). The measurements are used to evaluate the *size error* and the *SSIM error* as differences between the measured values and the desired ones.

We compare four different adaptation strategies, two from the original artifact [31] and two developed specifically for this work:

- **Random**: this adaptation strategy (from the original artifact) selects random encoding parameters. We use it as a baseline for our evaluation.
- **Model Predictive Control (MPC)**: this adaptation strategy (from the original artifact) exploits model predictive control algorithm [76]. It solves a model-based optimisation problem for each frame and uses the result to determine the encoding parameters for the next frame. For our tests, we used the tuning parameters from the original publication [73].
- **Integral**: we developed an heuristic adaptation strategy, inspired by control theory principles. Here, the size error is used to choose the quality parameter. If the size is larger than the desired one, the quality parameter is reduced by 5. If smaller, the quality is increased by 5. The SSIM index determines the choice of noise and sharpen filter radius. Both are increased by 1 if the quality is more than desired, and reduced otherwise. From an analytical perspective, the errors are *integrated* to perfect the encoding parameters choice.
- **ϵ-Greedy**: this adaptation strategy is based on the homonym machine-learning algorithm [77]. More specifically it belongs to the class of reinforcement learning algorithms. It alternatively leverages two adaptation approaches: (i) a *greedy* approach that exploits the knowledge of the best parameters already encountered with probability $1-\epsilon$, and (ii) a random approach that explores new possible choices, by randomly selecting new parameters with ϵ probability. The performance of a given choice of parameters is quantified based on the errors and normalised by the desired values. Higher similarity and lower size are desired, inducing errors that are close to zero. The greedy approach chooses the set of parameters that is associated to the lowest performance value. We use $\epsilon = 0.2$.

Test Design: In SAVE, adaptation takes place along a stream of frames, i.e. the feedback from one frame is used to improve the encoding of the next frame. To capture the behaviour of the adaptation strategy, each test should be an adequately long video, in which changes occur, triggering the need for adaptation. We would like to evaluate the performance of the different adaptation strategies independently from the content of the processed videos.

According to the proposed methodology, we define a set of videos that can be considered a random sample, with respect to their content. Here, we used the *User Generated Content* dataset from Youtube [78]. This dataset is representative of videos uploaded by users to Youtube. The videos are classified in categories and we focused on the sport category, because, due to the ever-changing scene, these are usually the most difficult to encode for real-time streaming and will expose the most of the adaptation strategy properties. The database contains 160 sport videos.

The adaptation strategy tries to achieve multiple objectives (a given size of the encoded frames, and a given content loss) at the same time. To capture the results obtained for both objectives, we define two different performance parameters, used to measure the outcome of the tests. The encoding performance on a single frame is directly quantified as the errors on: (i) the encoding size and (ii) the SSIM. For performance evaluation, we only consider relevant the cases in which the size is larger than the desired value or the quality is lower than the setpoint.

Intuitively, the size error is a problem when the images require more bytes than desired, and the SSIM quality is a problem when the image has less information than desired. We therefore evaluate the performance over a video of an adaptation strategy as the average of the size and SSIM errors weighted with the REctified Linear Unit, $relu(\cdot)$ function. The $relu(\cdot)$ function returns 0 for negative inputs and leaves the input unchanged for positive values. The complete formula for the performance parameters is shown in Equation (6), where $SSIM_v$ and $SIZE_v$ are the integrated errors on the video v, $SSIM_{sp}$ and $SIZE_{sp}$ are respectively the SSIM and size setpoints, $SSIM_i$ and $SIZE_i$ are the SSIM and size of the i-th frame and n_f is the number of frames in the video.

$$\begin{aligned} SSIM_v &= (1/n_f) \cdot \sum\nolimits_i relu(SSIM_{sp} - SSIM_i), \\ SIZE_v &= (1/n_f) \cdot \sum\nolimits_i relu(SIZE_i - SIZE_{sp}). \end{aligned} \tag{6}$$

In our evaluation, we use a SSIM reference of 0.9, preserving most of the content in the videos, and a frame size reference of 70% of the size of a frame randomly picked from the uncompressed video. The choice of having per-video references for the size is driven by the strong dependence of the frame size on the specific video.

Results: We ran the 160 encoding tests with each adaptation strategy. For each video v, we computed the two performance parameters $SSIM_v$ and $SIZE_v$ defined in Equation (6). The histograms in Figure 6 show the results of the tests.[5] The dashed grey lines mark the average performance

5. In the figure, we enforce the same scales for the axes to ease the comparison between the different plots. This results in hiding part of the plot of the size performance for the Integral strategy.

TABLE 1
SSIM performance [adimensional].

	Mean	Var	Max	EVT (30%)	EVT (20%)	EVT (10%)
Random	0.0710	±0.0054	0.3251	0.002720	0.002602	0.002264
MPC	0.1145	±0.0068	0.4565	0.004850	0.005141	0.002810
Integral	0.0315	±0.0029	0.1685	0.003230	0.002876	0.001808
Greedy	0.0135	±0.0018	0.1777	0.002010	0.003062	0.003528

TABLE 2
Size performance [bytes].

	Mean	Var	Max	EVT (30%)	EVT (20%)	EVT (10%)
Random	8806	±1033	82488	0.003925	0.005366	0.004677
MPC	492	±94	8718	0.006511	0.005625	0.005046
Integral	126373	±13942	992342	0.006526	0.006001	0.005460
Greedy	1885	±318	35191	0.004277	0.002834	0.004337

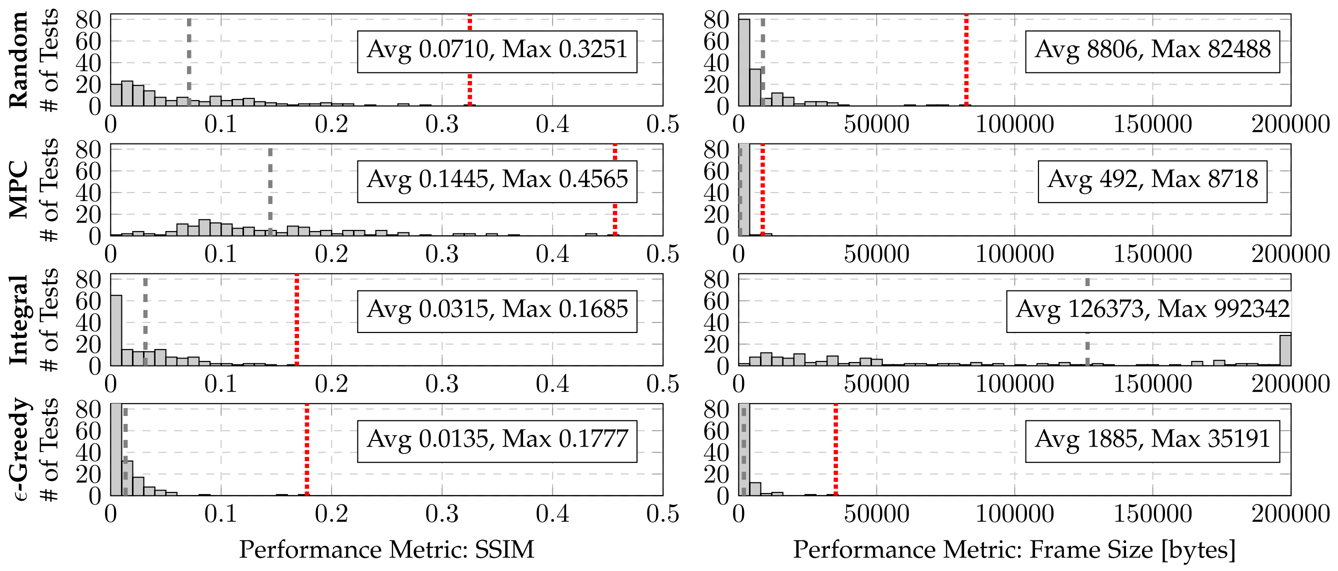

Fig. 6. SAVE: performance of the adaptation over the Youtube dataset with different techniques: Random, MPC, Integral, and ϵ-Greedy. The tables report the sampled mean and variance, the worst case and the worst case probability computed with EVT using different numbers of maxima. The histograms show the performance observed in all test cases and highlight the mean (grey dashed line) and worst case (red dotted line). Figure from [24].

for both similarity index and size, and the red dotted lines highlight the worst case experienced during the tests.

Tables 1 and 2 respectively show different performance metrics for the SSIM and frame size. The two leftmost columns contain the sampled mean and variance, used by the MC analysis. The *Max* column displays the maximum values experienced for the parameters in the tests, relevant for the ST approach. The three rightmost columns show three different probabilities computed with the EVT method. These are the probabilities of obtaining a performance value worse than the worse experienced value in the tests. We specifically computed three EVT probabilities using the same threshold value (the maximum value experienced in the experiments). The three probabilities correspond to the GEVD being fitted to respectively the 30%, 20%, and 10% largest values from the test outcomes. The chosen threshold value allows us to directly compare the results of the EVT approach with the ST probabilistic bound.

For what concerns the ST analysis, the number of performed tests $n = 160$ allows for the scenario parameters $\varepsilon = 0.03$ and $\beta = 0.008$. As for the TAS case study, this is not the only possible choice and a tighter bound could be traded for lower confidence (e.g. $\varepsilon = 0.01$ and $\beta = 0.04$) or vice versa (e.g. $\varepsilon = 0.05$ and $\beta = 0.0003$). Apparently, the two quantities hold equally for each of the tested adaptation strategies.

For the size performance, the MPC adaptation strategy vastly outperforms all the other strategies. This is achieved at the price of a SSIM adaptation performing worse than the Random strategy – i.e. the baseline. This is consistent with the adaptation objectives stated in the design of the strategy, where the size compression was considered the main objective [73]. This is equivalently observed by all the three alternative analysis techniques – i.e. comparing the first (MC) and third (EVT and ST) columns of the tables.

The Integral adaptation achieves the complementary result with respect to the MPC strategy. It presents good performance (among the strategies studied here) from the point of view of the SSIM but exposes the worse performance for what concerns the size. This can be attributed to the decoupled approach between the adaptation objectives pursued with this adaptation. Size and quality are not really decoupled (although the adaptation strategy treats them as such) and cannot effectively be treated separately.

When we compare the SSIM performance for the Integral and the ϵ-greedy strategies, the average and worst-case metrics are in slight disagreement. Whilst the former suggests a preference for the ϵ-greedy approach, the latter (the bare maximum) favours the Integral adaptation strategy. However, the tail of the histogram obtained with ϵ-

TABLE 3
Analysis of sampling bias for testing with SAVE. Frame size errors are normalised using the reference value, to allow a fair comparison between the different input sample sets.

	Video resolution	**360P**		**480P**		**720P**		**1080P**		**2160P**	
Strategy	Errors on	SSIM	Size	SSIM	Size	SSIM	Size	SSIM	Size	SSIM	Size
Random	Worst case	0.188	20468	0.325	22370	0.208	7743	0.262	82488	0.111	74245
	Average	0.070	1786	0.101	3084	0.058	3457	0.096	9418	0.031	25351
MPC	Worst case	0.250	6368	0.457	3044	0.312	799	0.435	8718	0.229	8577
	Average	0.142	254	0.177	209	0.129	165	0.186	603	0.091	1186
Integral	Worst case	0.139	44223	0.169	74377	0.098	173045	0.130	377724	0.067	992342
	Average	0.031	17714	0.044	27752	0.023	61233	0.046	162939	0.012	347691
Greedy	Worst case	0.035	846	0.178	13309	0.154	25153	0.054	35190	0.018	15967
	Average	0.011	265	0.023	1121	0.013	1487	0.017	2543	0.004	3852

greedy approach (Figure 6) seems lighter – i.e., less test cases performing "around and above" the performance value of 0.1. Intuitively, we would expect this to result in a higher probability of exceeding this bound. This is the probability that we computed with EVT in the right-most three columns of Table 1. Unfortunately, the value significantly depends on the number of maxima used for the GEVD fitting: if 30% of the values are considered maxima we should compare 0.32% for the integral adaptation and 0.2% for the ϵ-greedy strategy, otherwise 0.28% and 0.3%, or 0.18% and 0.35% if were respectively 20% and 10% of the values. Apparently, the conclusion on which is the best strategy will be different depending on the chosen number of maxima. Using EVT, we are not equipped with tools to select a number of maxima. Whilst these might seem minor variations in the probabilities, we recall that we are discussing probabilities of rare events (worst cases). Those probabilities are therefore intrinsically small and also minor variations can have high relative significance. This exposes one of the main limitations in applying the EVT to the testing of self-adaptive software. Conversely, ST assigns the same probability to the two adaptation strategies, leaving in some sense the final choice to the testing engineer. In this case, the sampled average from the MC method helps the testing engineer in choosing which adaptation strategy to favour. Despite this, it does not allow us to formally state that the ϵ-greedy approach outperforms the Integral strategy.

When simultaneously looking at both performance parameters, the machine-learning based approach achieves good performance. The SSIM performance is comparable to the one of the Integral adaptation and the size performance is in the order of the tens of kilobytes. This latter performance parameter can be considered small with respect to the biggest frames in the dataset, whose size is a few gigabytes. The ϵ-Greedy adaptation strategy proves therefore to be the best one at simultaneously achieving both adaptation objectives. This can be attributed to the exploration of the possible combinations of encoding parameters and the coupled feedback used for the two objectives.

Our testing methodology, when leveraging ST, guarantees that the comparison between the different adaptation strategies is fair. This is based on the rigorous quantification of the confidence we can have in obtained bounds. In particular, for the ϵ-Greedy algorithm, ST ensures that with a probability of $1-\varepsilon = 0.97$ we will not observe: (i) an error worse than 0.1777 for the SSIM performance parameter, and (ii) an error worse than 35191 Kb for the size performance parameter (see Equation 6 for the performance definitions). The confidence in our test campaign is of $1 - \beta = 0.992$, meaning that there is little probability of the choice of the adaptation strategy being wrong. Conversely, using MC and EVT, such formal statements would not be possible. Finally, if there was a need to tighten the bound or increase the confidence in the test campaign, the scenario theory would directly provide the extra number of test cases needed.

We highlight the difference between worst-case and average-case metrics. Analysing the average case (as done with the MC approaches) for the results in Figure 6, one would conclude that the Random adaptation strategy actually performs more or less as well as the others. However, this is not at all true for the worst-case metrics, which clearly expose the trade-off between size and quality and the difference between having an adaptation strategy that targets one or both these quantities and picking the next frame configurations at random.

Results with different inputs choices: To conclude the discussion on the results obtained with SAVE, we would like to discuss the impact of the choice of input videos. The discussion belongs to a more general remark on the choice of representative samples for the random inputs to be provided to the testing machinery.

Suppose that the broadcast videos are acquired by surveillance cameras and that we have a set of cameras with given resolutions. Initially, we envision cameras with 360P, 480P, 720P and 2160P resolutions. We therefore test the adaptation strategies using a diverse set of videos but only with the mentioned resolutions and draw some conclusion on the worst case frame size and similarity errors. When our system expands, we want to introduce additional cameras, with a new resolution of 1080P. In the testing phase, we did not collect collected any data on videos with such a resolution. However, given that we tested higher and lower resolutions, we could think that our test results are valid nonetheless and apply also to the extended set. We could then use the measured worst case in our tests and assume this is (most likely) not going to be violated. However, the input set of videos was not representative of the data that we then experience from the actual system.

To show this, we partition our initial data set in different subset with the given resolutions. Table 3 shows the worst case and average case errors for the subsets of our initial input sample. Computing the worst case value excluding videos from the 1080P resolution shows that it was necessary to test for this specific resolution and the corresponding test output changed our worst case. In fact, if one looks at the column representing the 1080P resolution videos, the worst case errors for the size of the resulting frames is much higher than it is for the other videos. Excluding the 1080P videos from the dataset would result is a much more favourable worst-case, which is not representative of what would have happened had the test sample being complete. On the contrary, including these tests from the beginning (i.e., before our system expansion) would have resulted in a conservative value being computed for the first setup.

The same remarks apply to average performance metrics and similar considerations hold for the video quality metric (SSIM) when one removes the 480P subset.

This shows that there is indeed a need for a representative set of input videos to properly provide guarantees on the worst-case experienced values. This does not simply apply to the video streaming service, but to any system that is tested using statistical methods (including ST).

5.3 Test Input Definition

Aim: The aim of this case study is to discuss how to define test inputs in a randomized testing campaign. We introduce and analyse an adaptive software application for traffic flow optimisation [32]. We assume that the software is utilised to understand if a given traffic adaptation strategy (implemented in the original artifact) is beneficial or not. We formally define the objectives of the testing campaign and show how those map to the choice of which inputs to randomise and which inputs have to be fixed across the tests (**CH4**). Leveraging ST we define the number of tests that are needed according to the desired probabilistic guarantees on the software performance (**CH1**). We ran two rounds of experiments (with and without adaptation) to identify the potential benefits of enabling the adaptation strategy. We discuss the connection between the outcome of the testing campaign and the choices made for the input randomisation. This case study shows the practical applicability of the proposed testing approach to the performance testing and evaluation of an adaptation strategy.

Self-Adaptive Software: TRAPP [32] is a self-adaptive framework for decentralized traffic optimization. It is based on the microscopic traffic simulator SUMO [79], and implements the interoperation with the decentralised combinatorial optimizer EPOS [80]. Within TRAPP, smart vehicles populate a simulated network of roads defined by the user. Vehicles can be introduced in the map in arbitrary points and will drive to reach a desired destination. Each vehicle produces different possible routes that it can take to reach its destination. The generated routes are associated to a specific cost that captures the preferences of the passengers. The routing options of every vehicle, together with their cost, are periodically collected. The EPOS optimiser is then executed to produce a route choice for each vehicle that accounts at the same time for both the desires of the car passengers and the overall efficiency of the road network. We extended the TRAPP artifact to enable repeated randomised testing.[6]

The traffic optimisation problem complexity grows exponentially with the number of cars, and achieving the global optimum in a general fashion is impossible. Moreover, the best approach to the optimisation problem depends on the current state of the network: e.g. the distribution of the cars in the streets, the specific destinations of the cars, and more. For this reason, adaptive approaches to the execution of the optimisation in EPOS have been proposed. The adaptation idea is to monitor in real-time the performance of the network and of the traffic flow optimisation. Leveraging this information, the EPOS optimisation can be adapted in order to improve the overall performance of the system. Among others, in TRAPP can be adapted the planning horizon, the planning fairness, or the agents selfishness.

In our case study, we assume a scenario in which the administration of a city wants to improve the city mobility by adopting the framework proposed by TRAPP. An adaptation strategy has been developed and needs to be tested in order to assess the potential benefit. More specifically, we consider the strategy *avoid-overloaded-streets* proposed in the original paper [32]. The idea is that the adaptive software layer monitors in real time which streets are closer to the limit of their capacity and adapts the EPOS optimisation so that those streets are avoided if possible.

Test Design: Analogously to the original paper [32], we quantify the performance of the adaptation strategy as the average trip overhead of the trips completed by all the cars. The trip overhead is defined as the ratio between the trip duration and the ideal trip duration that would be achieved in absence of other vehicles (hence if the vehicle was travelling always at the maximum allowed speed). A well performing adaptation strategy will be able to redirect the cars through the fastest route, thus reducing their traveling time and consequently the average trip overhead. Conversely, if the starting and end points of the cars are changed, the average trip is also likely to change. In this latter case, the performance change is related to the specific change in the testing scenario and not to the quality of the adaptation strategy. Leveraging the probabilistic approach taken in this paper, we therefore randomise the origin and destination of each trip over repeated tests. In this way, leveraging ST we obtain an evaluation of the adaptation performance that is independent of the source and destination parameters.

More in general, we depict a scenario in which the administration of the city mentioned above, wants a performance evaluation that is independent of both the specific vehicles and drivers that are currently populating the network, and also of the total number of vehicles in the streets. Following this specification, in our simulations we randomize these specific quantities. When a car is introduced in the network, we randomly pick values for its acceleration and deceleration (representing how performing are the engine and brakes of a car or driving style of the driver). Namely, for respectively the acceleration and deceleration

6. The code used for this set of experiments can be found in this repository: https://github.com/ManCla/TRAPP.

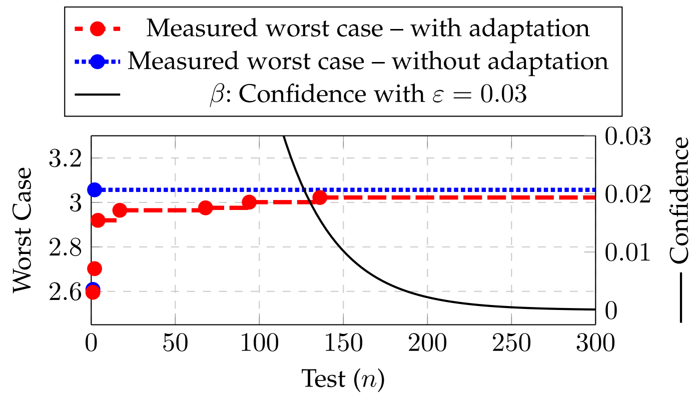

Fig. 7. TRAPP: Measured worst case with adaptation stratergy (in red), without adaptation (in blue), and confidence level (in black) according to the desired 97% probabilistic bound computed using Scenario Theory.

we used two truncated normal distributions $\mathcal{N}(4,2)$ and $\mathcal{N}(6,2)$, both with unit of measure [kmh/s]. In order to avoid unrealistic car performances both values are forced to be at least 1. Said distributions represent a statistical knowledge of the cars used in the city. To account for the preferences of the different drivers, we select the weights for the different routing options from a uniform distribution between 0 and 1. The different routing options can optimise the length of the trip, the expected average speed, or a combination of the two. To achieve independence from the specific trips, the cars are introduced in random points in the network and are supposed to drive to equally random points. Finally, we assume that the city administration has an evaluation of the possible number of cars that populate the network. Specifically, it has been estimated that the number of cars can be any value between 800 and 1200 with equal probability. Therefore, the number of cars included in each test is chosen according to the estimated distribution.

Conversely, in the testing of the adaptation strategy, it will not be relevant to randomize parameters that will be fixed once the system is deployed. Examples of such parameters are: the network, the triggering period of the adaptation, and the duration of each test. While randomising those parameters will provide a more general evaluation of the chosen adaptation strategy, it will not provide further information concerning the actual use of the adaptive system. For example, it is not important that an adaptation strategy performs well independently from the specific city where the TRAPP framework is implemented. Once the system is deployed, the network is not expected to change significantly and require a consequent reaction of the adaptive system. For what concerns instead the triggering period, its analysis should be systematic rather than randomised, so that an optimal choice can be made in the system design. This kind of analysis is out of the scope of this paper and has been discussed in recent related research [55], hence we chose the arbitrary period of 100 simulation ticks. The last mentioned parameter is the duration of the simulation. The choice of this parameter is driven by a trade-off between the efficiency and the relevance of the testing campaign. Apparently, efficiency purposes call for a simulation that is as short as possible. In our case study, we consider a test to have achieved significance when most cars have indicatively performed more than one trip. Since car trips can take from tens to hundreds of simulation ticks we chose for our tests a fixed value of 1000 simulation ticks.

In the depicted testing problem, we assume that a risk analysis requires that the obtained performance bound will hold in 97% of the cases. Equivalently, it is accepted a 3% probability that the adaptive system will not provide the expected performance: thus, for the application of ST ε is set to 0.03. Finally, we consider $1 - \beta = 0.9999$ and acceptable confidence in the final result – i.e. the probability that ε is effectively equal to 0.03 and not larger. By applying Equation 5 we obtain the number $n = 300$ of required tests. In order to evaluate the effectiveness of the proposed framework, we ran two separate round of tests: one including the proposed strategy and another one where the EPOS optimisation is never executed.

Results: Figure 7, shows the worst case performance observed along the two rounds of tests together with the confidence increase (quantified using ST). In the figure, the blue plot shows the worst case observed without adaptation, the red plot shows instead the worst case in presence of the adaptation. The tests on the adapted system showed an overall worst case of 3.0223 for the average trip overhead. The tests on the system without adaptation showed an overall worst case of 3.0568. Under the light of the chosen scenario parameters, we can state that, *with a confidence of* 99.99%, *there is a* 3% *probability that a combination of the randomised parameters will lead to an average trip overhead larger than* 3.0223 *and* 3.0568 *for respectively the adapted and non adapted cases*. Conversely, the obtained bounds will not hold if different choices are made for the fixed parameters: the network, the adaptation triggering period, and the test duration. If these latter parameters are changed, the tests would have to be performed again in order to obtain bounds that are valid for the new set-up.

The experiments did not show a significant difference in the performance of the network when the adaptation was introduced. Hence, it can be concluded that the proposed adaptation strategy does not provide any relevant contribution to the traffic flow, and the city administration should investigate different solutions. In fact, given the choices that were made for the random parameters, there is a 97% probability that this conclusion will hold for any number of cars between 800 and 1200. On the other side, if a different map was to be chosen, the obtained values wouldn't hold anymore and the obtained conclusion would not be valid anymore.

6 Limitations and Validity Threats

In this section, we discuss validity threats to the proposed approach. Validity threats can be divided into *internal* and *external*. The methodology of the paper is a direct application of ST [29] and as such does not pose any internal validity threat. On the contrary, we identify external validity threats in how the test inputs are collected, how the scenarios are randomised, and how many sample data are available. These external validity threats result in three main limitations of the proposed approach.

The first one is rooted in the definition of the testing of an adaptive system. The need for adaptation in a system rises from limited knowledge of the operational environment. This generates an intrinsic limitation to the definition of test cases, since the software, as a requirement, should adapt to new unforeseen circumstances. On the other side the testing process is only as effective as the test cases are representative of the real use case. These two objectives of the introduction of adaptation and rigorous definition of test cases are colliding [17]. The software engineer needs to synthesise a definition of the set of tests that adequately covers the adaptation use cases. However, the adaptive layer programmer has an interest in leaving the use cases as undefined as possible, for generality. In the TAS example, we would like the adaptation layer to handle general providers failures. However, this also means that (for proper testing) we need to define possible service failure patterns.

The second limitation arises from the interpretation of the performance parameters as random variables, and for this reason it is common to all the three statistical tools discussed. This interpretation is the key to exploit random sampling and to leverage the different theories that are based on probability theory. The roots of the limitation reside in the assumption of unbiased random sampling. Achieving unbiased random sampling can be challenging, especially when randomness cannot be quantified. The testing engineer must select a significant and relevant set of samples, e.g., sport videos with random content to test SAVE. The Scenario Theory reduces this limitation by not requiring any assumption on the probability distribution of the performance parameter. This allows to process data from tests that are conducted in a production environment (when available) and hence are, by definition, representative of the actual distributions.

A last limitation arises from the need to conduct many tests to achieve high confidence. EVT and ST are particularly affected by this, whilst MC seems to require a smaller number of tests – even though this cannot be generally guaranteed. Conducting many test cases can, in fact, be time-consuming and the process needs to be automated. On the other side, the number of needed tests is known a priory and allows for timely allocation of the resources. Also, within scenario theory the confidence grows exponentially with respect to the number of tests, avoiding the uncontrolled "explosion" of the number of tests to be executed.

7 Conclusions

In this paper we addressed the problem of testing the performance of a self-adaptive software system. Conventional testing techniques are limited in the guarantees they provide, due to the adaptation presence. The presence of adaptation makes this problem challenging, due to the need to test the system in the presence of uncertainty.

To deal with uncertainty, we investigated probabilistic techniques to analyse the resulting data. Moving to the probabilistic framework gave us the possibility of obtaining formal (albeit probabilistic) guarantees on the results of our testing campaign. We investigated classical statistical tools, like Monte Carlo Simulations and the Extreme Value Theory. In this investigation, we highlighted their limitations and shortcomings for the testing of adaptive software.

To overcome said limitations, we leveraged the scenario theory, a tool from robust control that was originally intended for the design of control systems in the presence of uncertainty. We reinterpreted the scenario theory results in light of our software testing problem. This allows us to provide formal probabilistic guarantees on the adaptation performance. Moreover, our method provides a probabilistic quantification of the testing adequacy, that can be used for the evaluation of testing coverage.

Finally, we empirically evaluated the effectiveness of our approach using three self-adaptive applications. We showed the trade-off between the experimental campaign volume and the confidence that can be obtained, demonstrated how to formally compare different adaptation strategies, and how to select randomised inputs for the testing process depending on the specified experimental evaluation objective. In our experimental results, we provided a thorough comparison of the application of Monte Carlo, Extreme Value Theory and the Scenario Theory. Our comparison showed why the latter is a better tool to test adaptive software.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH). The responsibility for the content remains with the authors.

References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, May 2009. [Online]. Available: https://doi.org/10.1145/1516533.1516538

[2] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1–26.

[3] B. H. C. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas, *Using Models at Runtime to Address Assurance for Self-Adaptive Systems*. Cham: Springer International Publishing, 2014, pp. 101–136. [Online]. Available: https://doi.org/10.1007/978-3-319-08915-7_4

[4] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. Lawrence, KS, USA: IEEE, 2011, pp. 283–292.

[5] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: A probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. Bergamo, Italy: ACM, 2015, pp. 1–12.

[6] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE. New York, NY, USA: ACM, 2014, pp. 299–310. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568272

[7] V. Gulisano, A. V. Papadopoulos, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Performance modeling of stream joins," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '17. New York, NY, USA: ACM, 2017, pp. 191–202. [Online]. Available: http://doi.acm.org/10.1145/3093742.3093923

[8] N. D'Ippolito, V. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel, "Hope for the best, prepare for the worst: Multi-tier control for adaptive systems," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 688–699. [Online]. Available: https://doi.org/10.1145/2568225.2568264

[9] S. Shevtsov and D. Weyns, "Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 229–241. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950301

[10] G. A. Moreno, A. V. Papadopoulos, K. Angelopoulos, J. Cámara, and B. Schmerl, "Comparing model-based predictive approaches to self-adaptation: Cobra and pla," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 42–53. [Online]. Available: https://doi.org/10.1109/SEAMS.2017.2

[11] D. Weyns, "Towards an integrated approach for validating qualities of self-adaptive systems," in *Proceedings of the Ninth International Workshop on Dynamic Analysis*, ser. WODA 2012. New York, NY, USA: Association for Computing Machinery, 2012, p. 24–29. [Online]. Available: https://doi.org/10.1145/2338966.2336803

[12] L. Briand, S. Nejati, M. Sabetzadeh, and D. Bianculli, "Testing the untestable: Model testing of complex software-intensive systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 789–792. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889212

[13] C. A. González, M. Varmazyar, S. Nejati, L. C. Briand, and Y. Isasi, "Enabling model testing of cyber-physical systems," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '18. New York, NY, USA: ACM, 2018, pp. 176–186. [Online]. Available: http://doi.acm.org/10.1145/3239372.3239409

[14] B. R. Siqueira, F. C. Ferrari, M. A. Serikawa, R. Menotti, and V. V. de Camargo, "Characterisation of challenges for testing of adaptive systems," in *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*, ser. SAST. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2993288.2993294

[15] A. Bertolino and P. Inverardi, *Changing Software in a Changing World: How to Test in Presence of Variability, Adaptation and Evolution?* Cham: Springer International Publishing, 2019, pp. 56–66. [Online]. Available: https://doi.org/10.1007/978-3-030-30985-5_5

[16] V. de Oliveira Neves, A. Bertolino, G. De Angelis, and L. Garcés, "Do we need new strategies for testing systems-of-systems?" in *Proceedings of the 6th International Workshop on Software Engineering for Systems-of-Systems*, ser. SESoS '18. New York, NY, USA: ACM, 2018, pp. 29–32. [Online]. Available: http://doi.acm.org/10.1145/3194754.3194758

[17] R. I. Bahar, U. Karpuzcu, and S. Misailovic, "Special session: Does approximation make testing harder (or easier)?" in *2019 IEEE 37th VLSI Test Symposium (VTS)*, April 2019, pp. 1–9.

[18] A. Bertolino, P. Inverardi, and H. Muccini, *Formal Methods in Testing Software Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 122–147. [Online]. Available: https://doi.org/10.1007/978-3-540-39800-4_7

[19] F. Munoz and B. Baudry, "Artificial table testing dynamically adaptive systems," *CoRR*, vol. abs/0903.0914, 2009. [Online]. Available: http://arxiv.org/abs/0903.0914

[20] T. Tse, S.-S. Yau, W. Chan, H. Lu, and T. Chen, "Testing context-sensitive middleware-based software applications," in *Proceedings - International Computer Software and Applications Conference*, vol. 1, 2004, pp. 458–466.

[21] K. Welsh and P. Sawyer, "Managing testing complexity in dynamically adaptive systems: A model-driven approach," in *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, April 2010, pp. 290–298.

[22] Z. Micskei, Z. Szatmári, J. Oláh, and I. Majzik, "A concept for testing robustness and safety of the context-aware behaviour of autonomous systems," in *Proceedings of the 6th KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications*, ser. KES-AMSTA'12. Berlin, Heidelberg: Springer-Verlag, 2012, p. 504–513. [Online]. Available: https://doi.org/10.1007/978-3-642-30947-2_55

[23] F. C. Ferrari, J. Noppen, R. Chitchyan, and A. R. Lancaster, "Investigating testing approaches for dynamically adaptive systems work in progress," in *Environment*, 2011.

[24] C. Mandrioli and M. Maggio, "Testing self-adaptive software with probabilistic guarantees on performance metrics," in *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. ACM, 2020.

[25] M. Böhme, "Assurance in software testing: A roadmap," in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 5–8. [Online]. Available: https://doi.org/10.1109/ICSE-NIER.2019.00010

[26] S. Dutta, W. Zhang, Z. Huang, and S. Misailovic, "Storm: Program reduction for testing and debugging probabilistic programming systems," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 729–739. [Online]. Available: https://doi.org/10.1145/3338906.3338972

[27] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2005.

[28] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*. AMLBook, 2012.

[29] G. C. Calafiore and M. C. Campi, "The scenario approach to robust control design," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 742–753, May 2006.

[30] D. Weyns and R. Calinescu, "Tele assistance: A self-adaptive service-based system examplar," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 88–92. [Online]. Available: http://dl.acm.org/citation.cfm?id=2821357.2821373

[31] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Self-adaptive video encoder: Comparison of multiple adaptation strategies made simple," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 123–128. [Online]. Available: https://doi.org/10.1109/SEAMS.2017.16

[32] I. Gerostathopoulos and E. Pournaras, "Trapped in traffic? a self-adaptive framework for decentralized traffic optimization," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2019, pp. 32–38.

[33] C. Martina Maggio, "Artifact esec/fse 2020," 2020. [Online]. Available: https://zenodo.org/record/3896795

[34] C. Trubiani and S. Apel, "Plus: Performance learning for uncertainty of software," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019, pp. 77–80.

[35] H. Wang, W. K. Chan, and T. H. Tse, "Improving the effectiveness of testing pervasive software via context diversity," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 2, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2620000

[36] Y. Qin, C. Xu, P. Yu, and J. Lu, "Sit: Sampling-based interactive testing for self-adaptive apps," *Journal of Systems and Software*, vol. 120, pp. 70 – 88, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121216301029

[37] I. d. S. Santos, "Testdas: Testing method for dynamically adaptive systems," Ph.D. dissertation, Universisdade Federal do Ceara, Fortaleza, Brazil, 2017.

[38] L. Yu, W. T. Tsai, Y. Jiang, and J. Gao, "Generating test cases for context-aware applications using bigraphs," in *2014 Eighth International Conference on Software Security and Reliability (SERE)*, June 2014, pp. 137–146.

[39] M. A. Mehmood, M. N. A. Khan, and W. Afzal, "Automating test data generation for testing context-aware applications," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, Nov 2018, pp. 104–108.

[40] J. Hänsel, T. Vogel, and H. Giese, "A testing scheme for self-adaptive software systems with architectural runtime models," in *2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, Sep. 2015, pp. 134–139.

[41] A. Reichstaller and A. Knapp, "Risk-based testing of self-adaptive systems using run-time predictions," in *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Sep. 2018, pp. 80–89.

[42] A. Arcuri, G. Fraser, and J. P. Galeotti, "Automated unit test generation for classes with environment dependencies," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 79–90. [Online]. Available: http://doi.acm.org/10.1145/2642937.2642986

[43] A. Hervieu, B. Baudry, and A. Gotlieb, "Managing execution environment variability during software testing: An industrial experience," in *Testing Software and Systems*, B. Nielsen and C. Weise, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 24–38.

[44] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The art of test case diversity," *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.02.022

[45] A. Arcuri and L. Briand, "Adaptive random testing: An illusion of effectiveness?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ser. ISSTA '11. New York, NY, USA: ACM, 2011, pp. 265–275. [Online]. Available: http://doi.acm.org/10.1145/2001420.2001452

[46] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, "The fuzzing book," in *The Fuzzing Book*. Saarland University, 2019, retrieved 2019-09-09 16:42:54+02:00. [Online]. Available: https://www.fuzzingbook.org/

[47] K. Yatoh, K. Sakamoto, F. Ishikawa, and S. Honiden, "Feedback-controlled random test generation," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 316–326. [Online]. Available: https://doi.org/10.1145/2771783.2771805

[48] P. Tramontana, D. Amalfitano, N. Amatucci, A. Memon, and A. R. Fasolino, "Developing and evaluating objective termination criteria for random testing," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 3, pp. 17:1–17:52, Jul. 2019. [Online]. Available: http://doi.acm.org/10.1145/3339836

[49] R. M. Hierons and M. G. Merayo, "Mutation testing from probabilistic and stochastic finite state machines," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1804–1818, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.06.030

[50] S. Rosario, A. Benveniste, S. Haar, and C. Jard, "Probabilistic qos and soft contracts for transaction-based web services orchestrations," *IEEE Transactions on Services Computing*, vol. 1, no. 4, pp. 187–200, Oct 2008.

[51] S.-Y. Hwang, H. Wang, J. Tang, and J. Srivastava, "A probabilistic approach to modeling and estimating the qos of web-services-based workflows," *Inf. Sci.*, vol. 177, no. 23, pp. 5484–5503, Dec. 2007. [Online]. Available: https://doi.org/10.1016/j.ins.2007.07.011

[52] K. Joshi, V. Fernando, and S. Misailovic, "Statistical algorithmic profiling for randomized approximate programs," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. IEEE Press, 2019, p. 608–618. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00071

[53] S. Dutta, O. Legunsen, Z. Huang, and S. Misailovic, "Testing probabilistic programming systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 574–586. [Online]. Available: https://doi.org/10.1145/3236024.3236057

[54] G. Canfora and M. Di Penta, "Testing services and service-centric systems: challenges and opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, March 2006.

[55] J. Dorn, S. Apel, and N. Siegmund, "Mastering uncertainty in performance estimations of configurable software systems," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 684–696. [Online]. Available: https://doi.org/10.1145/3324884.3416620

[56] C. P. Robert and G. Casella, *Monte Carlo Optimization*. New York, NY: Springer New York, 2010, pp. 125–165. [Online]. Available: https://doi.org/10.1007/978-1-4419-1576-4_5

[57] A. Jiménez-Martín, A. Mateos, and S. Ríos-Insua, "Monte carlo simulation techniques in a decision support system for group decision making," *Group Decision and Negotiation*, vol. 14, pp. 109–130, 01 2005.

[58] O. Johnson, *Information Theory and the Central Limit Theorem*. Imperial College Press, 2004. [Online]. Available: https://books.google.se/books?id=r5XI8a0lYykC

[59] B. Korver, "The monte carlo method and software reliability theory," 1994.

[60] H. Singh and P. Pal, "Software reliability testing using monte carlo methods," *International Journal of Computer Applications*, vol. 69, pp. 41–44, 05 2013.

[61] L. de Haan and A. Ferreira, *Extreme Value Theory: An Introduction (Springer Series in Operations Research and Financial Engineering)*, 1st ed. Springer, 2010.

[62] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart, "On the sustainability of the extreme value theory for wcet estimation," in *OpenAccess Series in Informatics*, vol. 39, 07 2014.

[63] F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella, "Upper-bounding Program Execution Time with Extreme Value Theory," in *13th International Workshop on Worst-Case Execution Time Analysis*, ser. OpenAccess Series in Informatics (OASIcs), C. Maiza, Ed., vol. 30. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 64–76. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4123

[64] R. Fisher, *The Genetical Theory of Natural Selection*. OUP Oxford, 1930.

[65] P. Embrechts, "Extreme value theory: Potential and limitations as an integrated risk management tool," *Derivatives Use, Trading and Regulation*, vol. 6, 02 2000.

[66] P. Embrechts, T. Mikosch, and C. Klüppelberg, *Modelling Extremal Events: For Insurance and Finance*. Berlin, Heidelberg: Springer-Verlag, 1997.

[67] B. Francis and P. Khargonekar, *Robust control theory*, ser. The IMA volumes in mathematics and its applications. Springer-Verlag, 1995. [Online]. Available: https://books.google.se/books?id=81vvAAAAMAAJ

[68] F. A. Ramponi and M. C. Campi, "Expected shortfall: Heuristics and certificates," *European Journal of Operational Research*, vol. 267, no. 3, pp. 1003 – 1013, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221717310330

[69] G. C. Calafiore, "Direct data-driven portfolio optimization with guaranteed shortfall probability," *Automatica*, vol. 49, no. 2, pp. 370 – 380, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109812005481

[70] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, *The Traveling Salesman Problem: A Computational Study*, ser. Princeton Series in Applied Mathematics. Princeton University Press, 2011. [Online]. Available: https://books.google.se/books?id=zfIm94nNqPoC

[71] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of web service compositions," *IET Software*, vol. 1, no. 6, pp. 219–232, December 2007.

[72] M. Caporuscio, R. Mirandola, and C. Trubiani, "Building design-time and run-time knowledge for qos-based component assembly," *Softw. Pract. Exper.*, vol. 47, no. 12, pp. 1905–1922, Dec. 2017. [Online]. Available: https://doi.org/10.1002/spe.2502

[73] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Automated control of multiple software goals using multiple actuators," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, 2017, pp. 373–384. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106247

[74] R. Edwards and N. Bencomo, "Desire: Further understanding nuances of degrees of satisfaction of non-functional requirements trade-off," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '18. New York, NY, USA: ACM, 2018, pp. 12–18. [Online]. Available: http://doi.acm.org/10.1145/3194133.3194142

[75] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[76] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice&mdash;a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, May 1989. [Online]. Available: http://dx.doi.org/10.1016/0005-1098(89)90002-2

[77] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[78] Y. Wang, S. Inguva, and B. Adsumilli, "Youtube ugc dataset for video compression research," 2019. [Online]. Available: https://media.withyoutube.com/

[79] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[80] E. Pournaras, P. Pilgerstorfer, and T. Asikis, "Decentralized collective learning for self-managed sharing economies," *ACM Trans. Auton. Adapt. Syst.*, vol. 13, no. 2, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3277668