STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **10**, February 2022, pp 147–159.
Published online in International Academic Press (www.IAPress.org)

An Adaptive Approach to Optimize Probabilistic Distributed Testing

Mohammed Amine Tajioue , Oussama Maakoul , Fatima Zahra Moutai , Sara Hsaini , Salma Azzouzi, My El Hassan Charaf*

Laboratory of Research in Informatics, Faculty of Sciences, Ibn Tofail University, Morocco

Abstract Typically, conformance testing consists of placing a set of parallel testers at each port of an implementation to ensure its conformance to the specification. However, a number of common fault detections occur if no coordination is made between these parallel testers and the implementation under test (IUT). Therefore, the test process must support mechanisms of coordination between these distributed components, particularly for implementation with stochastic behaviour. To this end, as well as to analyse the stochastic behaviour of the implementation under test, we propose in this paper an algorithm to generate for each tester a probabilistic local test sequence (PLTS) aiming to avoid both synchronization and observation issues. Finally, we suggest a new architecture based on Markov decision processes with an adaptive controller to control and optimize the whole testing process.

Keywords Probabilistic automata, Synchronization, Control, Observability, Optimization, Markov Decision Processes.

AMS 2010 subject classifications 68M14, 90C40, 94C12, 93E20.

DOI: 10.19139/soic-2310-5070-1205

1. Introduction

The latest developments in the field of distributed systems have led to an emphasis on testing aspects of these systems. Hence, an area referred by "Model Based Testing" (MBT) has emerged in which systems' testing is based either on models or on specifications. In the context of distributed testing, we usually consider the implementation under test (IUT) as a "black box" with some distributed interfaces, called ports and the behaviour of the IUT is known only by its interactions through these ports. In addition, we place a tester at each physical interface of the implementation (IUT) to perform our tests. However, the challenge lies in ensuring coordination between such testers and the implementation under test. In fact, several aspects such as the fault detections for example, could be significantly influenced by lack of coordination.

Overall, we face three main problems when testing distributed systems. These problems are known as controllability, observability and synchronization fault detections which are fundamental features of conformance distributed testing. These problems have a great influence on testing process of distributed testing systems, such as the execution of test sequences, the fault detectability of test system and the interpretation of testing results [1, 2, 3]. As solution to these problems, many works propose that testers exchange some messages through reliable communication channels [4] or to use eventually some timing constraints as explained in [5, 6].

On the other hand, previous works done on the distributed testing field have not widely addressed probabilistic models, even though distributed systems have often probabilistic features and characteristics. To this end, we use Probabilistic Automata (PAs) to define the probabilities to be associated with the events and particularly, we examine the properties used to specify the probability of exchanging messages among the various components of

*Correspondence to: Mohammed Amine Tajioue (Email: tajioue.mohammed.amine@gmail.com). Laboratory of Research in Informatics, Faculty of Sciences, Ibn Tofail University, Morocco.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

the distributed test system. In this perspective, the main contribution of this paper is to build Probabilistic Local Test Sequences (PLTS) using our algorithm. These sequences describe the events execution foreach tester and ensure coordination between testers without encountering observation and synchronization problems. Furthermore, we enhance our approach to control and optimize testing in distributed systems using an adaptive controller. In this case, the controller will learn the optimal policy from the specification, and make then the decision to exchange messages with the IUT implementation based on the learning process.

The paper is organized as follow: The section 2 describes the basic architecture for testing distributed systems and the problems encountered while testing these systems. Then, we review some related works in section3. Afterwards, we explain in the section 4 our algorithm to build the PLTS sequences according to the probabilistic model. Then, we give details on the testing procedure based on the novel local sequences (PLTS). In sections 5, 6 we explain the use of the Markov Decision Processes to control and optimize the distributed testing. Finally, some conclusions and future works are given in the last section.

2. Testing distributed systems

2.1. Architecture

The principle of the distributed test architecture is to coordinate a set of parallel test components (testers) with the implementation under test (IUT) using a communication service (Fig.1).

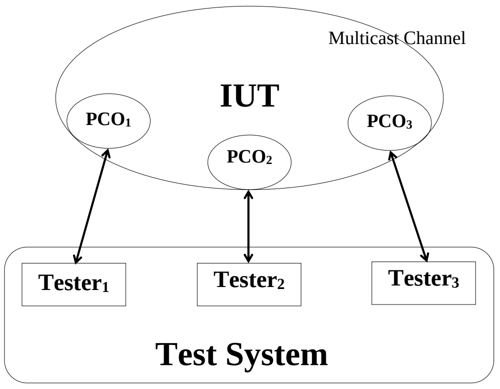

Figure 1. Distributed Testing Architecture.

In fact, the communication between testers and the IUT is performed through a set of ports called PCO (Port of Control and Observation). However, the testers communicate with each other using a multicast channel.

2.2. Distributed Testing problems

There are a number of issues that influence fault detection during the conformance testing process. In the main, three major problems known as controllability, observability and synchronization fault detections arise if no coordination ensured between the distributed test components:

• **Controllability problem**: refers to the test system ability to handle events at the corresponding ports PCOs in a given order. Indeed, the controllability problem arises while the Test system cannot guarantee that the implementation receives event of transition (i) before event of transition (i+1).

• **Observability problem**: It can be defined from Test System view as the capability of a Test System to observe the outputs of the IUT and decide which input is the cause of each output, it arises when two consecutive transition (i) and transition (i+1) occur on the same port k but only one of the transitions has an output in port k and the other one is an empty transition with no output. In this case the Test System cannot decide whether transition (i) or transition (i+1) is the cause of the output.

• **Synchronization problem**: In the distributed test, the testers will interact with the IUT through the communication channel. In order to reason about the synchronization problem and discuss the causality between the observed events, it is necessary to make assumptions regarding message latency.

In this paper, we aim to extend the distributed testing model on a probabilistic model by specifying the probabilistic behaviour. To this end, we review first some related works in the literature.

3. Literature review

Many works studied probabilistic testing of distributed systems. In this context, the authors in [7] proposed a framework to test formally systems with distributed ports where some choices are probabilistically quantified while other choices are non-deterministic. The main idea in [1, 2] is to construct a test sequence that causes no controllability or observability problems during its application in distributed test architecture. For some specifications, such sequence exists where the coordination is achieved via interactions with the IUT [8]. However, this case is not always true as detailed in [1, 9].

The authors in [10] suggest using multiple unique input/output (UIO) sequences to generate a test sequence and a method to minimize input/output operations as well as external coordination messages. Moreover, two model-based testing frameworks are explained in [11]. They additionally cover the stochastic aspects in hard and soft real-time systems. The authors use the stochastic automata as well as the theory of Markov automata for specifications, test cases, and conformance. Furthermore, the authors in [12] implement a new distribute cloud testing architecture for conformance testing to overcome problems occurred when testing distributed cloud. Thereafter, they extend their study to present in [13] a novel secure architecture-based conformance testing in order to check not only if the Cloud environment is conforming to its specification but also if respects some security policies using distributed testing rules.

The work [14] introduces the fundamental concepts of probabilistic bi-simulation via hypothesis testing whereas the authors in [15] show how to observe trace probabilities via hypothesis testing. Furthermore, the authors developed in [16] mutation testing techniques for probabilistic finite state machines (PFSMs) and probabilistic–stochastic finite state machines (PSFSMs), to produce test sequences that distinguish a PFSM M from a mutant M'. In addition, the aim of the statistical testing is to select entries for an IUT implementation in a probabilistic way in order to optimise some test measures as described in [17, 18, 19, 20]. Furthermore, an executable model-based testing framework for probabilistic systems is provided in [21]. The authors present therefore algorithms to automatically execute and assess test cases that are generated from a probabilistic specification.

Finally, this paper deals with the probabilistic properties of distributed testing and can be considered as a continuity of previous works [22, 23, 24]. In fact, the paper enhances our algorithms described in [25, 26, 27] for the generation of local test sequence. More precisely, the paper focus on the constraints that specify the probability of exchanging messages between testers as well as optimizing testing in distributed systems using an adaptive controller.

4. Probabilistic design: Overview

4.1. *Preliminaries*

In this section, we extend the definition of a probabilistic automaton with n-port to formally model and specify the stochastic behaviour of the distributed testing system.

For this purpose, we give some definitions of the n-ports probabilistic automata in distributed testing context as follows:

Definition1. We define a n-ports Probabilistic Automaton (np-PA) by A = (S, s_0, Act, T) with:

- S is a finite set of states;
- $s_0 \in$ S is the initial state;

- Act = $\Sigma \cup \Gamma$, is a finite set of actions with:
  i. $\Sigma = \{\Sigma_1, \Sigma_2, ..., \Sigma_n\}$ where Σ_i is a finite set of inputs of ports i, $\Sigma_i \cap \Sigma_j = \varnothing$ for i $\neq$ j and i,j = 1,2,...,n and $\Sigma = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_n$;
  ii. $\Gamma = \{\Gamma_1, \Gamma_2, ..., \Gamma_n\}$ where Γ_i is a finite set of outputs of ports i, $\Gamma_i \cap \Gamma_j = \varnothing$ for i $\neq$ j and i,j = 1,2,...,n and $\Gamma = \Gamma_1 \cup \Gamma_2 \cup ... \cup \Gamma_n$;
- T is a finite set of transitions and T $\subseteq$ S $\times$ Act $\times$ S $\times$ p[0,1].

Definition2. We define a transition as a tuple (s_1, σ, s_2, p[0, 1]) where:

- s_1, $s_2 \in$ S are the source state and destination state;
- $\sigma \in$ Act, is a sending of an input "x_i" or reception of an output "y_i" (figured respectively as !x_i and ?y_j);
- p[0,1]: For each T $\in$ Tr, we associate a probability p. Therefore, p is a non-negative real number $\leqslant$ 1 and the sum of the probabilities leaving a state: s $\in$ S is 1.

The meaning of T (s_1, σ, s_2, p) is that we can move from state s_1 to state s_2 with action $\sigma \in$ Act with probability p $\in$ [0,1].

Example:

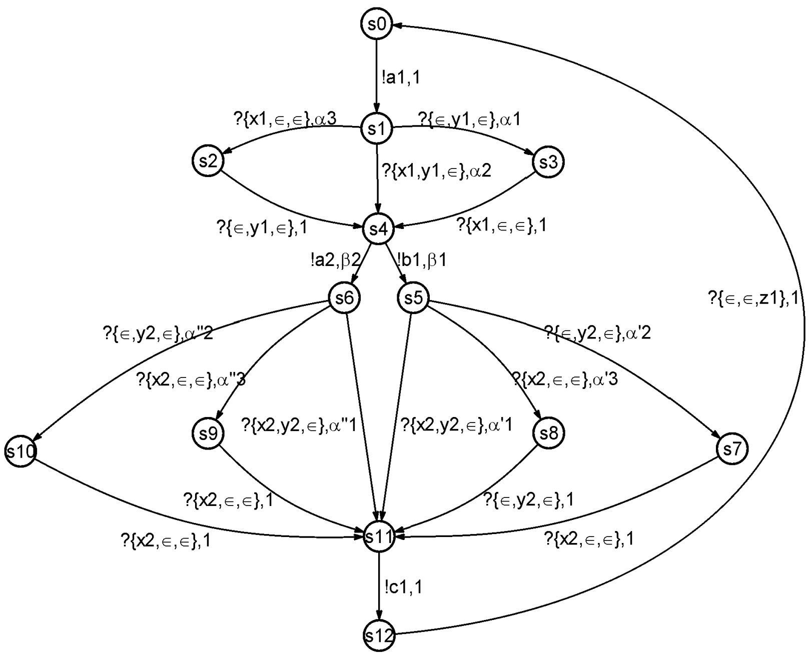

Figure 2. An example of a 3P-PA.

We give an example of a 3p-PA (Fig.2) where:

- S = {s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}};
- s_0 the initial state;
- Σ_1 = {a_1, a_2}, Γ_1 = {x_1, x_2},
- Σ_2 = {b_1}, Γ_2 = {y_1, y_2},
- Σ_3 = {c_1}, Γ_3 = {z_1};
- α_1, α_2 and α_3 such as α_1 + α_2 + α_3 = 1: are probabilities of transitions leaving s_1;
- α'_1, α'_2 and α'_3 such as α'_1 + α'_2 + α'_3 = 1: are probabilities of transitions leaving s_5;
- α''_1, α''_2 and α''_3 such as α''_1 + α''_2 + α''_3 = 1: are probabilities of transitions leaving s_6;
- β_1 and β_2 such as β_1 + β_2 = 1: are the probabilities of transitions leaving s_4.

4.2. PGTS: Probabilistic global test sequence

We define a probabilistic global test sequence (PGTS) as a test sequence of an np-PA corresponding to the sequence of transitions: Tr$_{!x1}$ .Tr$_{?y1}$ .Tr$_{!x2}$ .Tr$_{?y2}$ ...Tr$_{!xt}$ .Tr$_{?yt}$ where:

- Tr$_{!xi}$ is a transition of sending an input "x$_i$" in port-k. We denote: <!x$_i$, Prob(i)> where Prob(i) corresponds to the probability of sending an input;
- Tr$_{?yi}$ are the outputs received on different ports (1, 2, ..., j) with $1 \leqslant j \leqslant n$ resulting on sending an input "xi" to the IUT. We denote: {<?h$_1$, Prob(1)>, <?h$_2$, Prob(2)>, ..., <?h$_j$, Prob(j)>}.

The transitions corresponding to the example above (Fig.2) are expressed as follows:

- Tr$_{!a1}$ = (!a$_1$, 1);
- Tr$_{?y1}$ = ?((x$_1$, ϵ, ϵ), α_3).((ϵ, y$_1$, ϵ), 1);
- Tr$_{!b1}$ = (!b$_1$, β_1);
- Tr$_{?y2}$ = ?((x$_2$, y$_2$, ϵ), α'_1);
- Tr$_{!c1}$ = (!c$_1$, 1);
- Tr$_{?y3}$ = ?((ϵ, ϵ, z$_1$), 1).

Subsequently, we get an example of PGTS of 3p-PA given in Fig.2 as:

$$(!a_1, 1).?((x_1, \epsilon, \epsilon), \alpha_3).((\epsilon, y_1, \epsilon), 1).(!b_1, \beta_1).?((x_2, y_2, \epsilon), \alpha'_1).(!c_1, 1).?((\epsilon, \epsilon, z_1), 1); \tag{1}$$

4.3. PLTS: Probabilistic local test sequences

Generation of Probabilistic Local Test Sequences (PLTS): We suggest in this section our algorithm aiming to build the PLTS sequences related to each tester from a complete probabilistic global test sequence (PGTS) in order to ensure coordination and avoid observability and synchronisation issues.

A PLTS sequence has the form Tr$_1$ .Tr$_2$ .Tr$_3$ ....Tr$_n$ where each Tr$_i$ is either:

1. M$_i^k$Prob(i)^k if it is a message exchanged between tester-k and IUT. Each M$_i$ is:
   - !x$_i$: Tester-k sends "x$_i$" through port k to IUT or;
   - ?y$_i$: IUT sends "y$_i$" through port k to tester-k.
2. M$_i^k$ if it is a synchronization or observation message from a tester-k. Each M$_i$ is:
   - !O^k /?O^k: Observation message (sent to/received from) tester-k or;
   - !S^k /?S^k: Synchronization message S (sent to/received from) tester-k.

Therefore, each transition is considered as a data structure that contains the message to be exchanged as well as the probability of the event. We denote:

- 'Δ' is the symmetrical difference such as: AΔB = (A/B) $\cup$ (B/A) where '/' is the set difference;
- The function **Ports** provides the corresponding port for a given message and the function **Ports** is defined for a set y of messages by: Ports(y) = {k|$\exists$ a $\in$ y s.t. k = Port(a)}

Algorithm 1 An algorithm to build probabilistic local test sequences (PLTS).

Inputs

A (PGTS) Probabilistic Global Test Sequence w = w=Tr$_{!x1}$.Tr$_{?y1}$.Tr$_{!x2}$.Tr$_{?y2}$. ... .Tr$_{!xk}$.Tr$_{?yk}$

Outputs

PLTS : Probabilistic Local test sequences (W$_{p1}$, ...,W$_{pn}$)

Algorithm 1 An algorithm to build probabilistic local test sequences (PLTS).

```
1:  for k = 1, ...,n do
2:      $W_{pk} \leftarrow \epsilon$
3:  end for
4:  for i= 1, ...,t do
5:      k ← Port($x_i$)
6:      if i > 1 then
7:          SendTo ← (Port($y_i$) $\Delta$ Port($y_{i-1}$)) $\setminus\{k\}$
8:          if SendTo $\neq \varnothing$ then
9:              $W_{pk} \leftarrow W_{pk}.!O^{SendTo}$
10:             for all h $\in$ SendTo do
11:                 $W_{ph} \leftarrow W_{ph}.?O^{k}$
12:             end for
13:         end if
14:     end if
15:     $W_k \leftarrow W_k.!x_i$
16:     for all a $\in y_i$ do
17:         $Tr_{W_{?a}}$.Message ← ?a
18:         $Tr_{W_{?a}}$.Probability ← $Tr_{?a}$.Probability
19:         $W_{pk} \leftarrow W_{pk}.Tr_{W_{?a}}$
20:     end for
21:     if i < t then
22:         h ← Port($x_{i+1}$)
23:         if y = $\varnothing$ then
24:             $W_{pk} = W_{pk}.!S^{h}$
25:             $W_{ph} = W_{ph}.?S^{k}$
26:         else
27:             for all a $\in y_i$ do
28:                 if Port(a) $\neq$ h and Probability(a) $\neq$ 1 then
29:                     $W_p$.Port(a) ← $W_p$.Port(a).!$S^h$
30:                     $W_{ph} \leftarrow W_{ph}.?S^{Port(a)}$
31:                 end if
32:             end for
33:         end if
34:     end if
35: end for
```

By taking the global probabilistic test sequence defined in (1) as an example, the algorithm 1 above generates for each tester a local probabilistic test sequence (PLTS) as follows:

- Basically, the PLTS sequences are projections of the global probabilistic test sequence over the port alphabets;
- We add the reception of messages belonging to "y_i" to the appropriate sequences: The loop in (**line 16**);
- We add "?O^k" (k is port sending O) and "!O^k" (k is port receiving O) to the appropriate local test sequences (**lines 4 to 14**) in order to avoid observation problems. In fact, each tester receiving a message h $\in y_{i-1}$ should be able to determinate that h has been sent by IUT after IUT has received "x_{i-1}" and before IUT receives "x_i".
- We remind that the synchronization problem arises when we cannot guarantee that IUT will receive event of transition (i) before event of transition (i+1). To avoid Synchronization problem, we added "?S^k" (k is port sending S) and "!S^k" (k is port receiving S) to the appropriate local test sequences as follows:

– “$?S^k$” is added to w_{ph} where h is the tester sending “x_{i+1}” (**lines 25 and 30**);
– “$!S^k$” is added to the sequence of a tester receiving a message belonging to “y_i”, if $y_i \neq \varnothing$ (**lines 26 to 32**);
– if not “$!S^k$” is added to the sequence of the tester sending “x_i” (**lines 23 to 25**).

As a result, we get the following PLTS sequences by applying our algorithm 1 to the PGTS giving in (1). These PLTS sequences (W_{P1}, W_{P2} and W_{P3}) describe the behaviour of tester-1, tester-2 and tester-3 respectively:

$$\begin{cases} W_{p1} = !\{a_1, 1\}.?\{(x_1, \epsilon, \epsilon), \alpha_3\}.?\{x_2, \alpha'_1\}.!S^3.?O^3 \\ W_{p2} = ?\{(\epsilon, y_1, \epsilon), 1\}.!\{b_1, \beta_1\}.?\{y_2, \alpha'_1\}.!S^3.?O^3 \\ W_{p3} = ?S^1.?S^2.!\{c_1, 1\}.?\{z_1, 1\}.!O^{\{1,2\}} \end{cases}$$

Afterwards, each tester will execute the following procedure during the execution of its probabilistic local test sequence in order to get the test verdict:

- For each message “x_i” sending to the IUT or a synchronization/observation message, the tester supports the process of sending this message;
- If “M_i” is an expected message from the IUT or a synchronization/observation message, the tester waits for this message. If no message is received, or if the received message is not expected, the tester returns a verdict **Fail** (fail);
- If the tester reaches the end of its local test sequence, then it gives a verdict **Accept** (accepted). Thus, if all testers return a verdict **Accept**, then the test system ends the test with a global verdict **Accept**. Otherwise, the test system ends with **Fail** verdict.

In the next section, we introduce the Markov Decision Process to model the stochastic behaviour of the distributed testing system and to optimize the Test execution.

5. Test control and optimization

5.1. Optimal distributed test control architecture

The figure (Fig. 3) presents the Optimal Distributed Test Control (ODTC) architecture. An ODTC is composed of an adaptive controller connected to the testers, where each tester is attached to the IUT through the PCO ports.

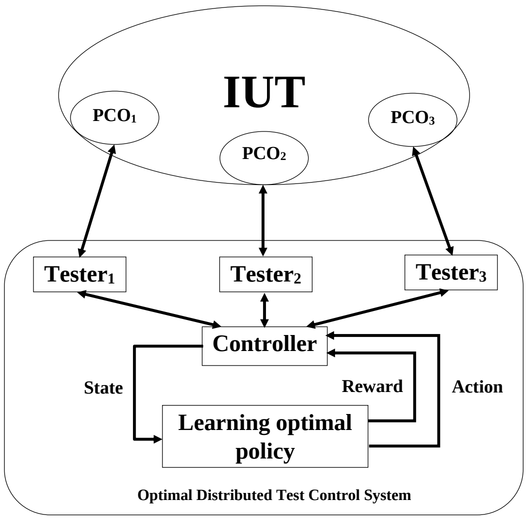

Figure 3. Optimal Distributed Test Control (ODTC) architecture.

To model the stochastic behaviour of our approach, we propose a formal model based on the MDP definition (Markov Decision Processes). Therefore, we extend some definitions related to the MDPs for inputs and outputs in distributed testing mode.

5.2. *Markov decision processes: Definitions*

Definition 3. A MDP is a 4-tuples (S, A, T, R) where:

- S is a finite set of states.
- $s_0 \in$ S is the initial state.
- A = $\Sigma \cup \Gamma$ is a finite set of actions, with:
  i. $\Sigma = \{\Sigma_1, \Sigma_2, ..., \Sigma_n\}$ where Σ_i is a finite set of inputs of ports i, $\Sigma_i \cap \Sigma_j = \varnothing$ for i $\neq$ j and i,j = 1,2,...,n and $\Sigma = \Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_n$;
  ii. $\Gamma = \{\Gamma_1, \Gamma_2, ..., \Gamma_n\}$ where Γ_i is a finite set of outputs of ports i, $\Gamma_i \cap \Gamma_j = \varnothing$ for i $\neq$ j and i,j = 1,2,...,n and $\Gamma = \Gamma_1 \cup \Gamma_2 \cup ... \cup \Gamma_n$;
- T : S × A × S $\in$ [0, 1] is a transition model. T(s, a, s') is the probability that if a controller performs action a$\in$ A when in state: s $\in$ S. Then, the state changes into state: s' $\in$ S. T is constrained such that the sum of the probabilities leaving a state: s $\in$ S with action: a $\in$ A is 1;
- R : S × A × S $\to \mathbb{R}$, is the immediate reward function, where R(s, a, s') is the immediate reward received after transitioning from state: s $\in$ S to state: s' $\in$ S, due to action a $\in$ A.

Definition 4. A policy π: S $\to$ A is a function that specifies the action π(s) $\in$ A that the controller will choose when in state: s $\in$ S and π^* denotes the optimal policy that optimizes the expected rewards.

Definition 5. A state value function V: S $\to \mathbb{R}$ induced by π is denoted V^π(s). V^π(s) is called the state value function for a given policy π. the value V^π(s) is the expected cumulative reward that a controller will receive if it follows the policy π when starting from state: s $\in$ S. We compute V^π(s) for each state: s $\in$ S as follows:

$$V^\pi(s) = R(s, \pi(s), s') + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s')$$

Where: $\gamma \in$ [0, 1] is the reward discount factor.

Definition 6. A state-action value function Q: S × A $\to \mathbb{R}$ induced for a given policy π is denoted Q^π(s, a). The value Q^π(s, a) is the expected cumulative reward that a controller will receive when starting in state: s $\in$ S, and executing the action a $\in$ A, then following the policy π. We compute Q^π(s, a) for each state s $\in$ S as:

$$Q^\pi(s, a) = R(s, a, s') + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s')$$

Definition 7. In a Markov decision process, an optimal policy π^* is the policy that maximizes the expected reward, and satisfying:

$$\pi^* \in arg \max_\pi V^\pi$$

According to the Bellman's equation [28], the corresponding optimal value function (denoted $V^* = V^{\pi^*}$) is defined as the best value that can be achieved in a state: s $\in$ S, where:

$$V^*(s) = \max_{a \in A} R(s, \pi(s), s') + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^*(s')$$

The corresponding optimal state-action value function is:

$$Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

$$V^*(s) = Q^*(s, \pi^*(s)) = \max_\pi Q^*(s, a)$$

6. Optimal control test process

The basic idea is to use a controller that will learn the optimal policy based on the specification of the (IUT) implementation.

6.1. Case Study

In this section, we propose a Markov Decision Processes (MDP) example to illustrate the stochastic behaviour of the distributed testing system. Therefore, we will use the MDP to model the synchronization and observability states that distributed testing system should avoid as well as rewards that the controller of the distributed testing system will apply in a given state if needed.

The figure (Fig. 4) gives an example of a MDP model of a 3p-IUT.

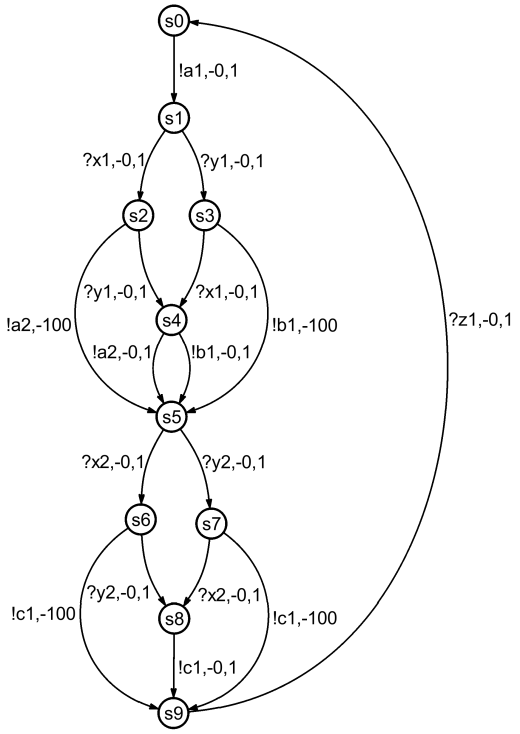

Figure 4. An example of a MDP model.

The example is defined with the following parameters:

- S = $\{s_0, s_1, ..., s_9\}$ a finite set of states;
- $s_0 \in$ S the initial state;
- A = $\{\Sigma_1, \Sigma_2, \Sigma_3, \Gamma_1, \Gamma_2, \Gamma_3\}$, such as:
  - $\Sigma_1 = \{a_1, a_2\}$;
  - $\Sigma_2 = \{b_1\}$;
  - $\Sigma_3 = \{c_1\}$;
  - $\Gamma_1 = \{x_1, x_2\}$;
  - $\Gamma_2 = \{y_1, y_2\}$;
  - $\Gamma_3 = \{z_1\}$;
  - R = -0.01, pen = -100 are the immediate rewards

6.2. Optimal control test Execution

In order to test the conformity of the IUT, the optimal control Test process is composed of two phases:

- The learning phase where the controller learns from the model based of the specification, to act optimally.
- The testing phase where the controller controls the distributed test based on the optimal policy learned in the first phase.

Therefore, the adaptive controller will learn the optimal policy from the specification, and based on the learning process the controller will make the decision to send an input "!x_i" to the IUT, or to wait for an output "?y_i" from the IUT.

6.3. Learning algorithm

In the learning process, the adaptive controller will apply the algorithm below to map the optimal policy based on the MDP model of the specification.

Afterwards, the adaptive controller will use the computed optimal policies to optimize and control the distributed testing process. In fact, the algorithm generates a deterministic policy from the model of the specification as follows:

- The algorithm takes as input the transition model T, the immediate reward function R, the discount factor γ, and the accuracy of estimation ϵ.
- We initialize the states values to 0 (**lines 1 to 3**) and we update then values for each state s $\in$ S by applying the Bellman equation to the function value iteratively (**lines 6 to 10**) until we converge to the optimal value (**lines 4, 5, 11**).
- We get as output a deterministic policy π that is the greedy policy with respect of the optimal value function for every state s $\in$ S.

Algorithm 2 Learning Optimal Policy.

Inputs

T is the transition model; R is the immediate reward function; γ the discount factor; ϵ the accuracy of estimation.

Outputs

a deterministic policy π, such that

$$\pi(s) = arg \max_{a \in A}\{R(s,a,s') + \gamma \sum_{s' \in S} T(s,a,s')V(s')\}$$

```
1:  for each s ∈ S do
2:      V(s) ← 0
3:  end for
4:  while Δ > ε do
5:      Δ ← 0
6:      for each s ∈ S do
7:          U(s) ← V(s)
8:          $V(s) \leftarrow \max_{a \in A} \{R(s,a,s') + \gamma \sum_{s' \in S} T(s,a,s')U(s')\}$
9:          $\Delta \leftarrow \max(\Delta, |U(s) - V(s)|)$
10:     end for
11: end while
```

After generating an optimal policy, the adaptive controller will learn to act optimally and choose the optimal action at each state in order to avoid observability and synchronization problems during distributed testing process.

The interaction between the IUT and the distributed testing system is considered asa decision making problem where the adaptive controller chooses actions based on the observed behaviour of the IUT and the optimal policy learned using the system specification.

An example of an optimal global control test sequence (OGCTS) deduced from the MDP model given in the figure (Fig.4) is:

$$!a_1.?y_1.?x_1.!b_1.?x_2.?y_2.!c_1.?z_1$$

And the local test sequences at each port is given as follows:

$$\begin{cases} W_{p1} = !a_1.?x_1.?x_2 \\ W_{p2} = ?y_1.!b_1.?y_2 \\ W_{p3} = !c_1.?z_1 \end{cases}$$

Therefore, the distributed testers will not need to exchange any kind of coordination messages as the adaptive controller will guarantee avoiding observability, and synchronization problems based on the optimal learning phase.

6.4. Test procedure

During the execution of the OGCTS, the adaptive controller will execute the following procedure:

- At each stage of the distributed testing process, the adaptive controller will choose the action to take based on the state of the IUT:
  - If it is a sending message to the IUT, the adaptive controller makes the corresponding tester supports the process of sending this message;
  - If it is an expected message from the IUT, the adaptive controller waits for this message from testers. if no message is received, or if the received message is not expected, the adaptive controller returns a verdict **Fail**.
- if the adaptive controller reaches the end of its OGCTS, then it gives a verdict **Accept**. Then the test system ends the test with a verdict **Accept**.

7. Discussion and Concluding Remarks

As a conclusion, we intended in this paper to use the core of probabilistic test theory to analyse the behaviour of the implementation under test. For this purpose, the paper focuses on the constraints that specify the probability of exchanging messages between testers as well as the mechanisms needed to optimize the distributed testing process. In this context, we propose an algorithm to generate probabilistic local test sequences (PLTS)in order to ensure coordination between testers without encountering observation and synchronization problems.

Compared to other previous studies that propose a model without a multicast channel between testers such as [7].The authors define indeed a weak implementation relationship based on probabilistic input and output transition systems. Our model, on the other hand, uses this channel to communicate messages between testers and subsequently ensure communication between testers to address the synchronization problem between testers. Moreover, the algorithm we proposed minimizes the messages exchanged between the testers, using only the observability and synchronization messages when needed. To achieve this, we used probabilistic automata to define the probabilities and model the messages exchanged between local testers and the IUT, allowing us to generate a probabilistic local test sequence (PLTS) for each tester by guaranteeing the absence of coordination, observability, and synchronization problems.

Furthermore, we suggest a new architecture based on Markov decision processes with an adaptive controller to control and optimize the testing process. The proposed model uses an adaptive approach to optimize the distributed testing process. This approach is based on Markov Decision Processes (MDP), where a controller is learning an optimal policy based on the MDP model of the specification by using the proposed algorithm. This approach will not only optimise the testing process by eliminating observability and synchronization messages between testers but also it will lead to an adaptive controller that will adapts to the probabilistic behaviour of the distributed testing system by learning to make the optimal decision during test execution.

As prospect, we plan to establish further investigations related to the consistency of our approach. To this end, we conduct actually more empirical studies to implement our algorithms and to evaluate the impact of the proposed approach for revealing errors in real case studies.

REFERENCES

1. K. C. Tai, and Y. C. Young, *Synchronizable test sequences of finite state machines*, Computer Networks, vol. 30, pp. 1111–1134, 1998.
2. R. M. Hierons, *Testing a distributed system: generating minimal synchronised test sequences that detect output-shifting faults*, Information and Software Technology, vol. 43, no. 9, pp. 551–560, 2001.
3. H. Ural, and D. Whittier, *Distributed testing without encountering controllability and observability problems*, Information Processing Letters, vol. 88, pp. 133-141, 2003.
4. O. Rafiq, and L. Cacciari, *Coordination algorithm for distributed testing*, The Journal of Supercomputing, vol. 24, pp. 203-211, 2003.
5. S. Azzouzi, M. Benattou, and M. E. H. Charaf, *A temporal agent-based approach for testing open distributed systems* Computer Standards & Interfaces, vol. 40, pp. 23– 33, 2015.
6. A. Khoumsi, *A new method for testing real time systems* Proceedings Seventh International Conference on Real-Time Computing Systems and Applications, pp. 441-450, 2000.
7. R. M. Hierons, and M. Núñez, *Implementation relations and probabilistic schedulers in the distributed test architecture*, Journal of Systems and Software, vol. 132, pp. 319–335, 2017.
8. G. Luo, R. Dssouli, G. v Bochmann, P. Venkataram, and A. Ghedamsi, *Test generation with respect to distributed interfaces*, Computer Standards & Interfaces, vol. 16, no. 2, pp. 119–132, 1994.
9. J. Chen, R. M. Hierons, and K. Ural, *Conditions for resolving observability problems in distributed testing*, Lecture Notes Computer Science (including Subseries Lecture Notes Artificial Intelligence Lecture Notes Bioinformatics), vol. 3235, pp. 229–242, 2004.
10. W. Y. Liu, H. W. Zeng, and H. K. Miao, *Multiple UIO-based test sequence generation for distributed systems*, Journal of Shanghai University, vol. 12, no. 5, pp. 438–443, 2008.
11. M. Gerhold, A. Hartmanns, and M. Stoelinga, *Model-based testing of stochastically timed systems*, Innovations in Systems and Software Engineering, vol. 15, no. 3-4, pp. 207–233, 2019.
12. F.-Z. Moutai, S. Hsaini, S. Azzouzi, and M. E. H. Charaf, *Testing Distributed Cloud: A Case Study*, 2019 International Symposium on Advanced Electrical and Communication Technologies (ISAECT), pp. 1–5, 2019.
13. F.-Z. Moutai, S. Hsaini, S. Azzouzi, and M. E. H. Charaf, *Security Testing Approach for IaaS Infrastructure*, Proceedings of the 2nd International Conference on Networking, Information Systems & Security (NISS19), pp. 1–5, 2019.
14. K. G. Larsen, and A. Skou, *Bisimulation through Probabilistic Testing (Preliminary Report)*, in Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (New York, NY, USA), pp. 344–352, Association for Computing Machinery, 1989.
15. L. Cheung, M. Stoelinga, and F. Vaandrager, *A testing scenario for probabilistic processes*, Journal of the ACM, vol. 54, no. 6, pp. 29–29, 2007.
16. R. M. Hierons, and M. G. Merayo, *Mutation testing from probabilistic and stochastic finite state machines*, Journal of Systems and Software, vol. 82, no. 11, pp. 1804– 1818, 2009.
17. H. Bohnenkamp, and A. Belinfante, *Timed testing with TorX*, Lecture Notes Computer Science, vol. 3582, pp. 173–188, 2005.
18. G. H. Walton, J. H. Poore, and C. J. Trammell, *Statistical testing of software based on a usage model*, Software: Practice and Experience, vol. 25, no. 1, pp. 97–108, 1995.
19. J. A. Whittaker, K. Rekab, and M. G. Thomason, *A Markov chain model for predicting the reliability of multi-build software*, Information and Software Technology, vol. 42, no. 12, pp. 889–894, 2000.
20. S. J. Prowell, *Computations for Markov Chain Usage Models*, Computer Science Technical Report UT-CS-03- 505, pp. 03-505, 2003.
21. M. Gerhold, and M. Stoelinga, *Model-based testing of probabilistic systems*, Formal Aspects of Computing, vol. 30, no. 1, pp. 77–106, 2018.
22. M. E. H. Charaf, M. Benattou, and A. S. A. Jess, *AGENT based architecture for testing distributed systems*, Journal of Information Science and Engineering, vol. 30, no. 5, pp. 1619–1634, 2014.
23. S. Hsaini, S. Azzouzi, and M. E. H. Charaf, *Testing Rules for Mapreduce Fameworks*, Colloquium in Information Science and Technology, 2018.
24. S. Azzouzi, S. Hsaini, and M. E. H. Charaf, *A Synchronized Test Control Execution Model of Distributed Systems*, International Journal of Grid and High Performance Computing, vol. 12, no. 1, pp. 1–17, 2020.
25. M. E. H. Charaf, and S. Azzouzi, *A colored petri-net model for control execution of distributed systems*, 4th International Conference on Control Decision and Information Technologies (CoDIT'17), pp. 277–282, 2017.
26. M. A. Tajioue, O. Maakoul, S. Hsaini, S. Azzouzi, and M. E. H. Charaf, *Towards Overcoming Issues of Testing Probabilistic Distributed systems*, 2020 7th International Conference on Control Decision and Information Technologies (CoDIT'20), vol. 1, pp. 903–907, 2021.

27. O. Maakoul, M. A. Tajioue, S. Hsaini, S. Azzouzi, and M. E. H. Charaf, *Testing Access Control List Policies in a Hadoop Environment*, 2020 7th International Conference on Control Decision and Information Technologies (CoDIT'20), vol. 1, pp. 831–836, 2021.
28. R. Bellman, *Dynamic Programming and Lagrange Multipliers*, Proceedings of the National Academy of Sciences, vol. 42, no. 10, pp. 767–769, 1956.