Recovery Process Visualization based on Automaton Construction

Hiroki Ikeuchi, Yosuke Takahashi, Kotaro Matsuda, Tsuyoshi Toyono
NTT Network Technology Laboratories, NTT Corporation, Tokyo 180-8585 Japan
Email: {hiroki.ikeuchi.re, yousuke.takahashi.bs, koutarou.matsuda.cz, tsuyoshi.toyono.ax}@hco.ntt.co.jp

***Abstract*—Since information and communications technology systems become large and complex, artificial intelligence (AI) technologies for automatic failure recovery have recently been developed. While AI-based failure recovery is promising, it raises an issue of interpretability, i.e., the problem in which operators cannot understand the target system behavior during the recovery process. To overcome this issue, we propose a method for constructing human-interpretable automata that describe the recovery process for each type of failure in accordance with the concept of proactive operation including chaos engineering. We define a new class of automata called observation-labeled finite automata (OLDFAs) with an observation function, which enable us to express recovery processes of the target system that can output high-dimensional data in a mathematical form. Our method constructs an OLDFA consistent with the target system on the basis of the L^* algorithm and clustering techniques. We also propose a method of creating a recovery process workflow (RPW) by merging OLDFAs for distinct failures. The RPW makes visible to operators what types of failures the target system is likely to currently have after specific actions have been taken and observations obtained. This would be beneficial in that operators can understand what AI-based operation takes the actions for and what states the target system is likely to be in during the recovery process. To evaluate the effectiveness of our methods, we carried out numerical simulations and experiments with container-based systems, showing our methods successfully construct automata and that the created RPW is interpretable.**

I. Introduction

Artificial intelligence (AI) technologies have been rapidly introduced to information and communications technology (ICT) system operation. As ICT systems grow more complex and the number of monitoring items increase, the conventional operation of manually handling observed data, determining actions, and reporting the procedures in a document becomes tremendously difficult. AI technologies can analyze a large amount of data produced from ICT systems and help operators make decisions. AI technologies are expected to facilitate the automatic operation of ICT systems with superhuman performance [1], [2].

Proactive AI learning toward automatic recovery is one of emerging fields to which AI technologies can greatly contribute. Reactive operation, which is an operation in which AI learns a model from data of failures that occurred in the past and prepares for the next occurrence, may be effective for typical failures. Such an operation, however, does not handle unforeseen failures or rare failures whose data cannot be gathered sufficiently. To overcome this difficulty, a proactive approach with intentional data generation has recently gained much attention. In proactive operation, by preparing a validation system and injecting many types of failures into it, operators can gather a massive amount of failure data in advance and train AI models for root cause analysis and automatic recovery. The concept of chaos engineering [3], which is a type of evacuation drill for systems conducted by injecting failures, has recently been proposed, and an increasing number of failure injection tools have become accessible. Some studies reported that proactive AI learning based on failure injection is empirically effective [4], [5]. Therefore, the fusion of AI learning and proactive operation has become promising.

AI-assisted recovery, especially that based on deep learning, raises the issue of interpretability. Since the inference mechanism of AI methods is largely a black box, operators often do not know the reason of AI's judgment. Moreover, in the middle of an automatic recovery process by AI, operators do not know where the system state is currently positioned in the whole recovery process, why the current action was executed, whether an alternative means of recovery exists, and so on. If the AI recovery fails in the middle of the process, operators need to recover from it manually, but it is difficult for them to understand the system state during the process.

Taking such problems into account, we address the problem of visualizing system behavior during the recovery processes. The recovery process is considered the state transition process in which we estimate the latent system state from observable metrics such as traffic or resource usage and execute a seemingly appropriate action, thereby moving the state transition toward a normal state. It is difficult to precisely understand and express the system state because observations randomly fluctuate with complex interactions between many processes working in the target system. By limiting the discussion only to the recovery process, however, the number of possible actions to take is around ten at most [6] and the recovery is completed by a few specific patterns of action sequences. This implies that the recovery process can be expressed as a sparse (i.e., only a few actions are meaningful), discrete (i.e., possible actions and transitionable states are finite), and deterministic (i.e., if the state and action are determined, the next state is also determined) state transition process. This assumption motivated us to take an approach of constructing a human-interpretable transition system, i.e., automaton, that represents a recovery process. In accordance with the idea of proactive operation, we also assume that injecting failures into the target system and executing various actions are possible. Under such assumptions, we consider how to construct interpretable automata representing recovery processes while interacting with the target system and obtaining the observations.

We have two technical challenges to overcome to achieve the above goals. The first one is how to extract and define the state in the recovery process. Although the states are assumed to be discrete, we do not know the number of states in advance and cannot directly associate states with observations because observations are typically multi-dimensional and stochastic. The second is that we need to also consider the situation in which failure candidates are not yet narrowed down uniquely. Even if we can construct an automaton for each type of failure in advance, in real failure handling there are times when operators have no idea what type of failure has occurred. In this case, operators do not know to which automaton should be referred. Therefore, we need to visualize a state transition considering failure-candidate uncertainty.

We propose a method of constructing automata representing the recovery process through proactive failure injection and data generation. To address the first challenge, we define a new class of automata, observation-labeled finite automata (OLDFAs) with an observation function, in which states are defined by both the observation distribution in the state and transitions from and to the state. The proposed method is basically based on the L^* algorithm developed by Angulin [7], which is a classical algorithm of constructing a deterministic finite automaton. Since the original L^* algorithm cannot address multi-dimensional and stochastic observations, we insert clustering procedures and attach discrete labels to each observation. By applying the L^* algorithm to such labeled states, we can appropriately define the states and transitions and construct OLDFAs. After constructing an OLDFA for each type of failure, to address the second challenge, we also propose a method of constructing a recovery process workflow (RPW), which enables operators to understand what states the system is likely to be in during the recovery process even if the type of failure that has occurred is unknown. We can easily construct an RPW by merging multiple OLDFAs for possible failures. The RPW improves the interpretability of automatic recovery in the sense that, in the case that an AI agent executes failure handling in practice, operators can follow its judgment and the recovery process by referring to only the RPW.

The rest of this paper is organized as follows. In Section II, we introduce related work. In Section III, we describe two problems to be considered and define the OLDFA and RPW. In Section IV, we explain the observation table, which plays a central role in our methods. In Section V, we describe the two proposed methods, corresponding to the two problems. In Section VI, we discuss the evaluation of our methods through numerical simulations and experiments, showing their feasibility and effectiveness. We conclude this paper in Section VII.

II. Related work

Automaton construction An automaton is a mathematical model for describing the system behavior in response to given sequences and has a wide variety of applications in modern society. Angluin proposed the L^* algorithm [7], which enables us to create minimum consistent automata while issuing a polynomial number of queries to the target system, and its extensions and applications have been explored extensively [8], [9]. An automaton is also considered a tool of providing human-interpretability [10] and has gained attention in AI and machine learning. By extending the L^* algorithm, methods of constructing deterministic finite automata (DFAs) [11], weighted finite automata (WFAs) [12], and probabilistic DFAs (PDFAs) [13] from recursive neural networks (RNNs) have recently been proposed. Unlike these studies that intended to directly give the interpretability to AI models, our study aimed to model recovery processes of failed ICT systems through automaton-construction methods.

Recovery process visualization There have been studies [6], [14] on visualizing the recovery process as a workflow chart that indicates in what order what actions should be taken, on the basis of trouble tickets and product documentations. Although these studies and ours had similar goals, ours did not require prepared data but dynamically constructing automata while interacting with the target system.

Recovery process modeling Conventionally, the recovery process has been expressed as several types of probabilistic models. Specifically, the partially Markov decision process (POMDP) and its simplified variants are often used [15]–[17] due to their high expressive power. However, the POMDP-based formulation usually requires ad hoc prior distributions and large amount of training data. Given the fact that recent ICT systems produce complexly correlated observations, our OLDFA modeling is considered more computationally efficient and has less room for the bias of artificial design.

Automatic recovery To achieve automatic recovery, we need to develop two components: a mechanism of preparing many types of failed environments and a method for learning an optimal policy of recovering from such failures. The latter has been tackled using AI techniques such as reinforcement learning [4], [18], [19], in which an agent automatically learns an optimal recovery policy by executing various actions in a failed environment. The former has rapidly grown with the concept of proactive operation, including chaos engineering [3], which enables us to collect failure data and understand the behavior of failed systems in advance and facilitates the research of recovery policy learning. Our proposed methods also rely on such a failure injection mechanism. On the other hand, ours play a complimentary role to recovery-policy-learning methods. That is, they are aimed at creating a recovery policy that works without human intervention, while ours are aimed at extracting system behavior and presenting a human-interpretable workflow to operators.

III. Problem statement

Figure 1 illustrates the two problems we consider: (i) OLDFA construction and (ii) RPW construction. Problem (i) is formulated as follows. We assume having a target system X in which we can attempt various actions and restore to the initial

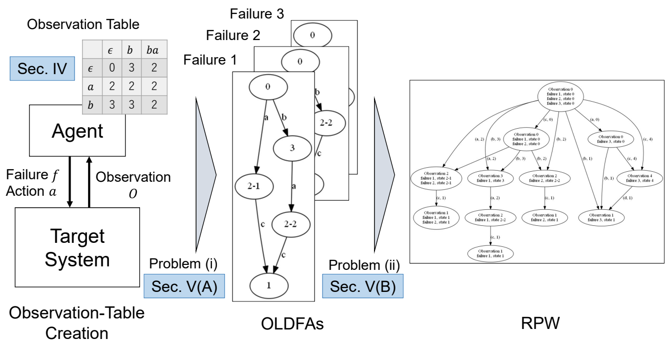


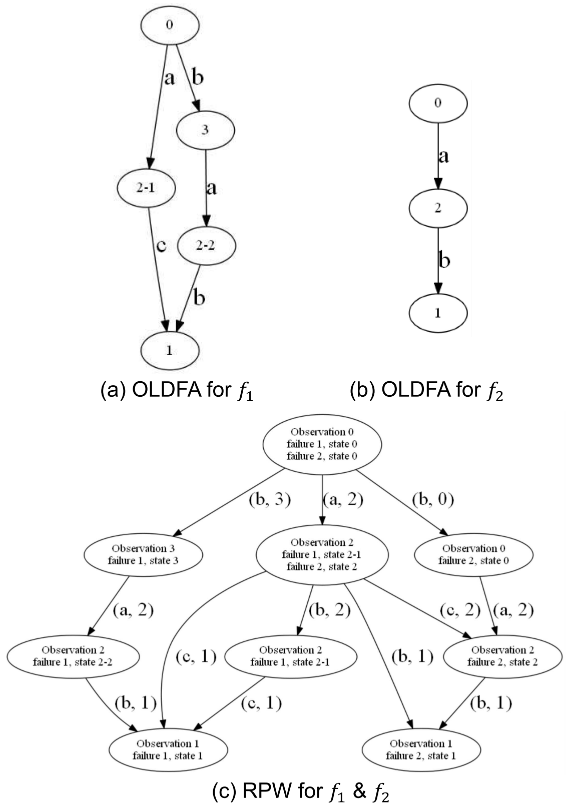
Fig. 1: Overview of problems

state any number of times, where X may be a validation system or emulator that behaves similarly to the product system. We inject (the programmed agent injects) a certain failure f into X, for example, using chaos engineering tools, and execute various recovery actions. We then obtain observation sequences as the state of X transits with the actions. Our goal is constructing a type of DFA, an OLDFA, from which humans can interpret the behavior of X by appropriately defining the states and transitions on the basis of the sequences of the executed actions and obtained observations. We repeat this procedure as f changes. In problem (ii), we aim at constructing a workflow, RPW, which indicates which type of failure X is likely to have and what states it is likely to be in. This can improve the interpretability of system behavior during the real-failure-handling process. In what follows, we formally define the OLDFA and RPW and describe them with examples.

We assume that the system behavior under f, denoted as $X(f)$, can be represented as an *OLDFA* with an *observation function*. Formally, $X(f)$ is a tuple $\langle Q, A, \delta, q_0, \{F_l\}_{l\in L}, \{O_l\}_{l\in L}\rangle$, where Q is a non-empty finite set of states, A is a non-empty set of actions, $\delta : Q \times A \to Q$ is a transition function, $q_0 \in Q$ is the initial state, $F_l \subset Q$ is a subset of Q whose elements have label $l \in L = \{0, 1, \cdots, |L| - 1\}$, and O_l is an observation function defined as an $\mathbb{R}^n$-valued random variable ($n \in \mathbb{N}$ is arbitrary and fixed). Let A^* be the set of sequences over A and $\epsilon \in A^*$ be the 0-length action sequence. The domain of δ can be extended to $Q\times A^*$ as $\delta(q, \epsilon) = q, \delta(q, u\cdot a) = \delta(\delta(q, u), a)$ for $q \in Q, a \in A$, and $u \in A^*$, where $\cdot$ denotes the concatenation of actions. Each $q \in Q$ must belong to just one of $\{F_l\}_{l\in L}$, meaning that it holds that $Q = \cup_{i=0}^{|L|-1} F_i$ and $F_i \cap F_j = \emptyset$ $(i \neq j)$. The initial state $q_0 \in F_0$ represents a state just after f is injected, and the states labeled with $l = 1$, i.e., $\{q\}_{q\in F_1}$, are the recovered states, conventionally called the accepting states. For simplicity, we assume that $|F_1| = 1$ and we have a means to judge whether the system is normal. A state labeled with l emits an observation value following a random variable O_l $(l \in L)$ with the corresponding label l. Note that two distinct states q_1 and q_2 belonging to a common F_l output observations following the same random variable O_l. This reflects the fact that even if one executes a certain action (e.g., submits an updated configuration) and the system changes its state, the observation does not always change explicitly (e.g., the observation does not change until the equipment is rebooted and the renewed configuration is reflected). In other words, the Markov property does not hold if we focus only on the observation. We define a label function $label : Q \to L$ such as $label(q) = l$ if and only if $q \in F_l$ for $q \in Q$.

We cannot directly observe either Q, δ, or $\{F_l\}_l$. All we can do is take actions through trial and error and obtain the corresponding observations. Namely, we execute $u = a_1 a_2 \cdots a_{|u|} \in A^*$ and obtain a realization of $O(u) \equiv O_{label(\delta(q_0,u))}$ (and observations in the middle of the execution of u). What we consider in problem (i) is how to construct an OLDFA of $X(f)$ and what action sequences should be executed to do so.

Example 1. Figure 2 (a) shows an OLDFA in which $Q = \{q_0 = 0, 1, \text{2-1}, \text{2-2}, 3\}, A = \{a, b, c\}, L = \{0, 1, 2, 3\}$, $F_0 = \{0\}, F_1 = \{1\}, F_2 = \{\text{2-1}, \text{2-2}\}, F_3 = \{3\}$. States are expressed as nodes with their label numbers and δ as arrows with actions. Self-loop transitions are not shown for simplicity. Distinct states 2-1 and 2-2 have the same label 2, and thus they emit observations obeying the same random variable O_2. The action sequences *abac* and *bab* can recover from failure, i.e., $\delta(0, abac) = \delta(0, bab) = 1$ (note that $\delta(\text{2-1}, a) = \delta(\text{2-1}, b) = \text{2-1}$), while *bac* cannot as $\delta(0, bac) = \text{2-2}$. □


0
a
b
3
2-1
a
c
2-2
b
1
0
a
2
b
1
Observation 0 failure 1, state 0 failure 2, state 0
(b, 3)
(a, 2)
(b, 0)
Observation 3 failure 1, state 3
Observation 2 failure 1, state 2-1 failure 2, state 2
Observation 0 failure 2, state 0
(a, 2)
(b, 2)
(c, 2)
(a, 2)
(c, 1)
(b, 1)
Observation 2 failure 1, state 2-2
Observation 2 failure 1, state 2-1
Observation 2 failure 2, state 2
(b, 1)
(c, 1)
(b, 1)
Observation 1 failure 1, state 1
Observation 1 failure 2, state 1

(a) OLDFA for f_1 (b) OLDFA for f_2

(c) RPW for f_1 & f_2

Fig. 2: Examples 1 and 2

We now explain the RPW. OLDFAs are assumed to have already been constructed for each f. When the failure type cannot be identified with the initial observation, i.e., O_0, however, operators do not know what type of failure is currently occurring and to which automaton should be referred. Thus, we need to merge the automata for all types of possible failures and create the RPW showing us what types of failures are still possible and what states the system is likely to be in under specific actions executed and observations obtained.

Given N OLDFAs $M^i = \langle Q^i, A, \delta^i, \{F^i_l\}_{l\in L^i}, \{O^i_l\}_{l\in L^i}\rangle$, each corresponding to failure f_i ($i = 1, 2, \cdots, N$), their *RPW* is defined as a directed graph with nodes having the information on label and possible states for each possible failure type and directed edges having the information on action and next label. Formally, a node is expressed as a tuple $\langle l, \Theta\rangle$, where $l \in \cup_{i=1}^N L^i$ is a label and $\Theta = \left(\theta^1, \cdots, \theta^N\right) \in \left(Q^1 \cup \{\text{N/A}\}\right) \times \cdots \times \left(Q^N \cup \{\text{N/A}\}\right)$ denotes a possible state with label l for each f_i. In other words, a node $\langle l, (\theta^i)_i\rangle$ enumerates candidate states that emit an observation labeled with l under the condition that each failure has occurred. If $\theta^i = \text{N/A}$, failure f_i is impossible. A directed edge labeled with a pair $\left(a \in A, l \in \cup_{i=1}^N L^i\right)$ from node $u = \langle l_u, (\theta^i_u)_i\rangle$ to $v = \langle l_v, (\theta^i_v)_i\rangle$ exists if and only if for i s.t. $\theta^i_u \neq \text{N/A}$, $\delta^i(\theta^i_u, a) = \theta^i_v$, and for i s.t. $\theta^i_u = \text{N/A}$, $\theta^i_v = \text{N/A}$, and $l = l_v$. Without getting into more formal discussion, we show an example of the RPW below.

Example 2. See Figure 2. The RPW in (c) is created from the OLDFAs for (a) f_1 and (b) f_2. Self-loop transitions are not shown. The very top node in the RPW has $l = 0$, $\Theta = (\theta^1, \theta^2) = (0, 0)$. If b is executed and observation labeled with $l = 3$ is obtained, we transit to the node in which $l = 3$, $\Theta = (3, \text{N/A})$, meaning that we successfully narrow down the candidates of failure and only f_1 is possible. The subsequent actions and observations $(a, 2)$ and $(b, 1)$ lead to recovery from f_1. □

IV. Preliminaries

Our OLDFA construction method is based on an algorithm inspired by the L^* algorithm [7]. In the original L^* algorithm, one submits queries systematically and expands what is called an observation table to construct a DFA on the basis of queries' two-valued answers, i.e., "accepted"(1) or "rejected"(0). The observation table for DFAs is straightforwardly extended to OLDFAs. In this section, we explain the observation table for OLDFAs along with our problem setting.

Given A and L, an *observation table* is a tuple $T = \langle P, S, h\rangle$, where $P \subset A^*$ is a non-empty prefix-closed set, $S \subset A^*$ is a non-empty suffix-closed set, and $h : (P \cup P \cdot A) \cdot S \to L$ is a function. A set $E \subset A^*$ is called *prefix-closed* (*suffix-closed*) if and only if for $u \in A^*$ and $a \in A$, $u \cdot a \in E \Rightarrow u \in E$ ($a \cdot u \in E \Rightarrow u \in E$). By definition, a non-empty prefix-closed (suffix-closed) set has $\epsilon \in A^*$. An observation table can also be represented as a two-dimensional matrix with rows labeled with elements of $P \cup P \cdot A$, columns labeled with elements of S, and entries defined as $\tilde{h} : (P \cup P \cdot A) \times S \ni (p, s) \mapsto h(p \cdot s) \in L$. We define $\tilde{\mathbf{h}} : (P \cup P \cdot A) \to L^{|S|}$ as $\tilde{\mathbf{h}}(p) = (h(p, s_1), h(p, s_2), \cdots, h(p, s_{|S|}))$, where $p \in (P \cup P \cdot A)$ and $s_j \in S$ for $j = 1, 2, \cdots, |S|$. That is, $\tilde{\mathbf{h}}(p)$ expresses the row vector indexed with $p \in (P \cup P \cdot A)$.

The observation table $\tilde{h}$ is called *closed* if and only if for each $p' \in P \cdot A$ there exists $p \in P$ such that $\tilde{\mathbf{h}}(p') = \tilde{\mathbf{h}}(p)$. The $\tilde{h}$ is called *consistent* if and only if for every $p_1, p_2 \in P$ and $a \in A$, it holds that $\tilde{\mathbf{h}}(p_1) = \tilde{\mathbf{h}}(p_2) \Rightarrow \tilde{\mathbf{h}}(p_1 \cdot a) = \tilde{\mathbf{h}}(p_2 \cdot a)$.

Once the closed and consistent $\tilde{h}$ is created, the consistent OLDFA $M(P, S, h)$, i.e., the OLDFA satisfying $\delta(q_0, p \cdot s) = \tilde{h}(p, s)$, is trivially constructed as $Q \equiv \{\tilde{\mathbf{h}}(p) \mid p \in P\}$, $q_0 \equiv \tilde{\mathbf{h}}(\epsilon)$, $\delta(\tilde{\mathbf{h}}(p), a) = \tilde{h}(p{\cdot}a)$, and $F_c \equiv \{\tilde{\mathbf{h}}(p) \mid p \in P, h(p) = l\}$. Note that the closedness and consistency of the observation table makes $M(P, S, h)$ well-defined. That is, the closedness guarantees the existence of the destination of any transition from any state, and the consistency guarantees that the transition from the identical state with the identical action is deterministic. This construction is optimal in terms of automaton size, as described in the following theorem.

Theorem 1. Given the closed and consistent observation table $\langle P, S, h\rangle$, $M(P, S, h)$ is the minimum OLDFA consistent with $\langle P, S, h\rangle$. That is, other OLDFAs consistent with $\langle P, S, h\rangle$ must have a strictly greater number of states than $M(P, S, h)$.

Sketch of proof. As in a previous study [7], the consistency of $M(P, S, h)$ with $\langle P, S, h\rangle$ can be shown by induction with respect to the length of P and S. Moreover, letting the number of states of $M(P, S, h)$ be n, we can show that OLDFAs consistent with $\langle P, S, h\rangle$ must have at least n states and that OLDFAs with just n states are isomorphic to $M(P, S, h)$. □

Example 3. See Table If, which is a closed and consistent observation table. The upper six rows are $\tilde{\mathbf{h}}(p)$ for $p \in P$ and the lower 12 rows $\tilde{\mathbf{h}}(p)$ for $p \in P \cdot A$. The corresponding OLDFA is shown in Figure 3a. Row $\tilde{\mathbf{h}}(\epsilon)$ corresponds to state 0, $\tilde{\mathbf{h}}(a)$ to 2-1, $\tilde{\mathbf{h}}(b)$ to 3, $\tilde{\mathbf{h}}(aa)$ to 1, and $\tilde{\mathbf{h}}(ab)$ to 2-2. Although states 2-1 and 2-2 have the same label, they are distinguished by the destinations of the transition caused by b. □

V. Proposed methods

In the following two subsections, we describe the proposed methods of OLDFA and RPW construction, respectively.

A. OLDFA construction

We assume that the system behavior under failure f, $X(f)$, follows a latent OLDFA with an observation function. Our basic idea of creating an observation table T is accordance with the L^* algorithm. However, unlike the original setting of the L^* algorithm, we need to deal with the multi-dimensional stochastic observation. Thus, we insert the clustering steps in the middle of the expansion of T and attach a temporary label to each observation result.

The outline of the algorithm for constructing an OLDFA is summarized in Algorithm 1, which is almost the same as the L^* algorithm except for the Expand-table procedure. In this algorithm, the following two steps are repeated until the constructed OLDFA passes the equivalence test. In the first step, we check the closedness and consistency of T under construction. If either is broken, we extend P or S and expand the table in accordance with the procedure Expand-table, which is explained later. After obtaining the closed and consistent T, in the second step, we create a consistent OLDFA in the manner described in Section IV and execute the equivalence test, namely, check whether the OLDFA really represents $X(f)$. Specifically, we inject f into X, randomly sample action sequences, execute them in X, and obtain the observations. We then check whether the labels of the

Algorithm 1 OLDFA construction

```
Initialize:
Observation table $T = \langle P \leftarrow \{\epsilon\}, S \leftarrow \{\epsilon\}, h\rangle$
Observation data $Data$={}
EXPAND-TABLE($P, S, T$)

$learned \leftarrow False$
while not $learned$ do
    ### Checking the closedness and consistency ###
    while $T$ is not closed or not consistent do
        if $T$ is not closed, i.e., $\exists p^* \in P \cdot A$ s.t. $\tilde{\mathbf{h}}(p^*) \neq \tilde{\mathbf{h}}(p)$ for $\forall p \in P$ then
            $P \leftarrow P \cup \{p^*\}$
            EXPAND-TABLE($\{p^*\}, S, T$)
        if $T$ is not consistent, i.e., $\exists p_1, p_2 \in P, a \in A$, and $s \in S$ s.t. $\tilde{\mathbf{h}}(p_1) = \tilde{\mathbf{h}}(p_2), \tilde{h}(p_1 \cdot a \cdot s) \neq \tilde{h}(p_2 \cdot a \cdot s)$ then
            $S \leftarrow \{a \cdot s\} \cup S$
            EXPAND-TABLE($P, \{a \cdot s\}, T$)

    ### Equivalence test ###
    Create the consistent OLDFA $M(P, S, h)$
    $learned \leftarrow True$
    for $i = 1, 2, \cdots, N$ do
        Sample $u \in A^*$
        if $\delta(q_0, u) \neq h(u)$ then
            $U \leftarrow \{u\} \cup$ ($u$'s prefixes)
            $P \leftarrow P \cup U$
            EXPAND-TABLE($U, S, T$)
            $learned \leftarrow False$
            break
Return $M(P, S, h)$
```

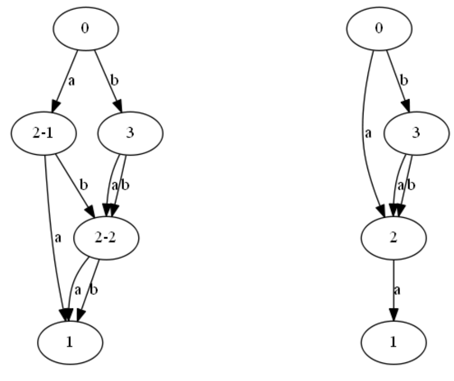
observations and those which are outputted by the OLDFA are consistent. If so, the algorithm outputs the OLDFA and halts. If not, we expand T with the found counterexample and start the first step over again.

In the middle of the algorithm, we expand T in accordance with Procedure 2. In this procedure, we execute the action sequences made by connecting the elements of P' and S' and obtain the observations $\{O(p, s)\}_{p \in P', s \in S'}$. Since they are multi-dimensional, we cluster them along with the past observations and attach the labels $\tilde{h}(p \cdot s)$ to them. By doing this, we can expand T with the rows $p \in P'$ and columns $s \in S'$. We should use a clustering method that has the following two properties, (i) the number of clusters does not need to be given and (ii) specific distributions of data, such as Gaussian, are not assumed. These properties reflect the fact that we do not know the number of states in advance and do not know the distributions of observations, respectively. Taking them into account, we used DBSCAN [20] along with dimension reduction as a clustering method in the evaluation part.

Example 4. For simplicity, we consider the situation in which the clustering follows the ground truth and labels attached by clustering are unchanged once attached before no matter how many times the clustering is repeated. We assume that $X(f)$ is expressed as the OLDFA in Figure 3a. Our algorithm expands T, as in Table I. After executing actions in $A = \{a, b\}$ and attaching labels to their observations by clustering, we find that T is not closed because $\tilde{\mathbf{h}}(\epsilon) \neq \tilde{\mathbf{h}}(a)$, as in Table Ia. Thus, we add a to P and repeat observation and clustering. We then find that T is not closed because there is no $p \in P$ such that $\tilde{\mathbf{h}}(p) = \tilde{\mathbf{h}}(b)$, as in Table Ib. Thus, we add b to P. In the same manner, we add aa to P in Table Ic. Since the T in Table Id is closed and consistent, the OLDFA expressed as in Figure 3b is returned. Through an equivalent test, however, we obtain a counterexample abb, which leads to an observation labeled with 2 in the estimated OLDFA, while $X(f)$ returns a different observation labeled with 1. Thus, we add abb and its prefix ab to P. In Table Ie, we find that T is not consistent because $\tilde{\mathbf{h}}(a) = \tilde{\mathbf{h}}(ab)$ but $\tilde{\mathbf{h}}(ab) \neq \tilde{\mathbf{h}}(abb)$. Since the suffix b can distinguish two states $\delta(q_0, a)$ and $\delta(q_0, ab)$, we add b to S and expand T. We obtain closed and consistent T, as in Table If, and the corresponding OLDFA in Figure 3a. Note that both $\delta(q_0, a)$ and $\delta(q_0, ab)$ are in F_2 but distinguished as states 2-1 and 2-2. This OLDFA finally passes the equivalence test, and the algorithm stops. □

Procedure 2 EXPAND-TABLE

```
Input: $(P', S', T)$
for $p$ in $P'$ do
    for $s$ in $S'$ do
        Execute $p \cdot s$ on $X$
        $Data(p \cdot s) \leftarrow O(p \cdot s)$
Cluster observation data in $Data$ and attach labels $\tilde{h}(p, s)$ to elements of $T$
```



$$\begin{tikzcd}
 & 0 \arrow[dl, "a"'] \arrow[d, "b"] \\
2\text{-}1 \arrow[dr, "b"] \arrow[ddr, "a"'] & 3 \arrow[d, "a/b"] \\
 & 2\text{-}2 \arrow[d, "a/b"] \\
 & 1
\end{tikzcd}$$

(a) OLDFA for Table If

$$\begin{tikzcd}
0 \arrow[dd, bend right, "a"'] \arrow[dr, "b"] & \\
 & 3 \arrow[dl, "a/b"] \\
2 \arrow[d, "a"] & \\
1 &
\end{tikzcd}$$

(b) OLDFA for Table Id

Fig. 3: Example 3

B. Recovery process workflow

Given multiple OLDFAs, we first need to make labels in different OLDFAs consistent. That is, when we create OLDFAs independently, labels between two OLDFAs are not associated. Thus, by using a unified clustering or classification mapping from observations to labels, we standardize the meaning of labels between OLDFAs. This can also be done by constructing only one OLDFA with multiple initial states using the algorithm described in Section V-A and extracting

TABLE I: Example of observation table expansion

(a) Not closed: $\tilde{\mathbf{h}}(p \in P) \neq \tilde{\mathbf{h}}(a)$

S / P	ϵ
ϵ	0
$\underline{\boldsymbol{a}}$	$\underline{\mathbf{2}}$
b	3

(b) Not closed: $\tilde{\mathbf{h}}(p \in P) \neq \tilde{\mathbf{h}}(b)$

S / P	ϵ
ϵ	0
a	2
a	2
$\underline{\boldsymbol{b}}$	$\underline{\mathbf{3}}$
aa	1
ab	2

(c) Not closed: $\tilde{\mathbf{h}}(p \in P) \neq \tilde{\mathbf{h}}(aa)$

S / P	ϵ
ϵ	0
a	2
b	3
a	2
b	3
$\underline{\boldsymbol{aa}}$	$\underline{\mathbf{1}}$
ab	2
ba	2
bb	2

(d) Return OLDFA in Figure 3b

S / P	ϵ
ϵ	0
a	2
b	3
aa	1
a	2
b	3
aa	1
ab	2
ba	2
bb	2
aaa	1
aab	1

(e) Not consistent: $\tilde{\mathbf{h}}(a) = \tilde{\mathbf{h}}(ab)$, $\tilde{\mathbf{h}}(ab) \neq \tilde{\mathbf{h}}(abb)$

S / P	ϵ
ϵ	0
$\underline{\boldsymbol{a}}$	$\underline{\mathbf{2}}$
b	3
aa	1
$\underline{\boldsymbol{ab}}$	$\underline{\mathbf{2}}$
abb	1
a	2
b	3
aa	1
$\underline{\boldsymbol{ab}}$	$\underline{\mathbf{2}}$
ba	2
bb	2
aaa	1
aab	1
aba	1
$\underline{\boldsymbol{abb}}$	$\underline{\mathbf{1}}$
$abba$	1
$abbb$	1

(f) Return OLDFA in Figure 3a

S / P	ϵ	b
ϵ	0	3
a	2	2
b	3	2
aa	1	1
ab	2	1
abb	1	1
a	2	2
b	3	2
aa	1	1
ab	2	1
ba	2	1
bb	2	1
aaa	1	1
aab	1	1
aba	1	1
abb	1	1
$abba$	1	1
$abbb$	1	1

the maximum components reachable from each initial state, which is an OLDFA for each failure.

Once label-consistent OLDFAs M^i $(i = 1, \cdots, N)$ are obtained, the RPW is straightforwardly constructed by defining the nodes and directed edges while checking the destination of transition for each q_i and a. We summarize the algorithm of RPW construction in Algorithm 3, which is implemented with breath-first construction for node $u = \langle c, \Theta \rangle$. The returned *Edges* is a set of directed edges, each consisting of the source node, destination node, and its label (a, c). By connecting these directed edges, we finally obtain the RPW.

VI. Evaluation

We conducted numerical simulations and experiments to verify the effectiveness and feasibility of our methods. In the numerical simulations, we evaluated the OLDFA-construction method quantitatively, while in the experiments, we qualitatively evaluated the effectiveness of the RPW-construction method using a Kubernetes system as a case study.

A. Numerical simulations

1) Settings: We prepared three types of ground-truth automata shown in Figure 4. Automaton (B) has three distinct states that have the same label, while (C) has arrows of opposite directions, which indicate rollback actions. They are typical recovery processes that can be seen in real failure handling. In each state, the 20-dimensional data are emitted

Algorithm 3 RPW construction

```
Input: $M^i = \langle Q^i, A, \delta^i, \{F^i_l\}_l, \{O^i_l\}_l \rangle$ $(i = 1, \cdots, N)$
Initialize:
Set $Edges \leftarrow \emptyset$
Queue $nodeQueue \leftarrow \left[\left\langle l = 0, \Theta = \left(q^1_0, \cdots, q^N_0\right)\right\rangle\right]$
Set $seen \leftarrow \left\{\left\langle l = 0, \Theta = \left(q^1_0, \cdots, q^N_0\right)\right\rangle\right\}$

while $nodeQueue \neq \emptyset$ do
  $u = \langle l, \Theta \rangle \leftarrow nodeQueue$.get()
  for $a \in A$ do
    $\tilde{\Theta}_l = (\text{N/A}, \cdots \text{N/A}),\ (l \in \cup_i L_i)$
    for $i \in \{1, \cdots, N\}$ do
      if $\Theta[i] \neq$ N/A then
        $\tilde{l} \leftarrow label(\delta^i(\Theta[i], a))$
        $\tilde{\Theta}_{\tilde{l}}[i] \leftarrow \delta^i(\Theta[i], a)$
    for $l \in \cup_i L_i$ do
      $v = \left\langle l, \tilde{\Theta}_l \right\rangle$
      $Edges$.add($(u, v, (a, l))$)
      if $v \notin seen$ then
        $nodeQueue$.put($v$)
        $seen$.add($v$)
Return $Edges$
```

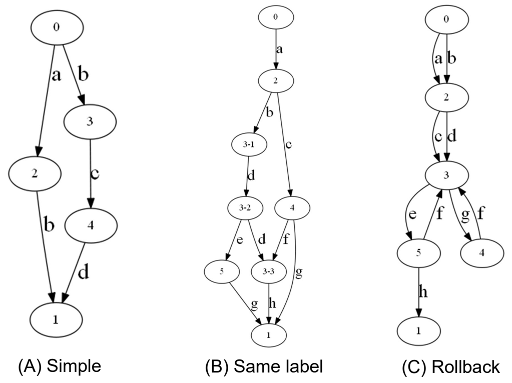


Fig. 4: Ground truth OLDFA

corresponding to its label in accordance with an isotropic Gaussian distribution whose mean is chosen uniformly randomly in $[0, 100]^{20}$ and standard deviation is 10. In the clustering procedure, we first mapped the 20-dimensional data into a two-dimensional space by using UMAP [21], a dimension reduction technique, and then conducted clustering by using DBSCAN [20]. Since our problem setting is new and no comparison method exists, we implemented the L^* algorithm [7] as a baseline. Although the L^* algorithm does not handle observations but whether the state is accepted, the obtained DFA is always correct if a sufficient number of samples of action sequences are given in the equivalence tests. Thus, by comparing our OLDFA-construction method with the L^* algorithm, we can investigate how our clustering procedures affect the accuracy of automaton construction and how many actions can be reduced with our method until the construction of the automaton is completed. We also changed the number of samples of the equivalence test to 50, 100, and 200, and investigated its effect. We carried out 100 runs for each setting

TABLE II: Simulation results of average Acc (± standard error), ARI(n), and OTS (± standard deviation) for each setting

Automaton type	# of equivalent	Acc		ARI (n = 4, 6, 8)		OTS	
	test samples	baseline	proposed	baseline	proposed	baseline	proposed
(A) Simple	50	1.00(0.00)	0.99(0.01)	1.00 1.00 1.00	1.00 1.00 1.00	332.3(59.3)	163.0(54.9)
	100	1.00(0.00)	1.00(0.00)	1.00 1.00 1.00	1.00 1.00 1.00	346.9(54.5)	172.7(54.0)
	200	1.00(0.00)	1.00(0.00)	1.00 1.00 1.00	1.00 1.00 1.00	347.2(50.2)	163.1(50.0)
(B) Same label	50	0.42(0.05)	0.59(0.05)	1.00 0.99 0.97	1.00 1.00 0.98	1122.4(464.5)	662.2(215.9)
	100	0.63(0.05)	0.84(0.04)	1.00 1.00 0.99	1.00 1.00 1.00	1402.1(365.7)	757.9(198.1)
	200	0.79(0.04)	0.92(0.03)	1.00 1.00 1.00	1.00 1.00 1.00	1529.7(305.6)	710.9(138.4)
(C) Rollback	50	0.22(0.04)	0.33(0.05)	0.97 0.91 0.84	0.98 0.93 0.87	376.4(166.1)	158.9(68.6)
	100	0.32(0.05)	0.44(0.05)	0.97 0.92 0.86	0.98 0.94 0.88	409.0(178.0)	170.4(74.4)
	200	0.60(0.05)	0.65(0.05)	0.98 0.96 0.92	0.99 0.96 0.92	520.6(195.1)	193.2(80.8)

and averaged the evaluation measures. Note that both the L^* algorithm and our algorithm depend on the randomness of the equivalence test samples and our algorithm depends also on the randomness of observations.

2) Evaluation Measures: We evaluated the results in accordance with the following three measures.

Accuracy of construction (Acc): We judge whether the estimated (OL)DFA is isomorphic to the ground-truth (OL)DFA. The Acc is the ratio of the number of such cases to 100 runs.

Adjusted rand index score (ARI(n)): This shows the degree of how similarly the obtained OLDFA describes the ground truth, as defined below. We consider all the action sequences $\{u\}$ with length n and execute them in the estimated DFA and ground-truth DFA. By considering destination states as clustering labels (note that this "label" is not equal to $l \in L$), we obtain two clustering results of the estimated DFA and ground-truth DFA, i.e., $\{\delta^{est}(q_0^{est}, u)\}_u$ and $\{\delta^{GT}(q_0^{GT}, u)\}_u$. We finally calculate the adjusted rand index [22], a well-known measure of the similarity of clusterings, for two clustering results $\{\delta^{est}(q_0^{est}, u)\}_u$ and $\{\delta^{GT}(q_0^{GT}, u)\}_u$ to obtain the value of ARI(n). In other words, ARI(n) quantifies the similarity of the estimated and true DFAs regarding whether two distinct action sequences lead to the same state. ARI(n) takes a value between $[0, 1]$, and the closer to 1 ARI(n) is, the more similar to the ground truth the estimation is.

Observation table size (OTS): This is defined by the number of elements of T, which indicates the number of action sequences required until the construction of the (OL)DFA is completed. The smaller OTS is, the less time the (OL)DFA construction requires.

3) Results: We summarize the simulation results in Table II. For type (A), both the baseline and our OLDFA-construction method achieved around 1 for Acc and ARI. Compared to the baseline, our method successfully reduced the number of required actions by nearly half. This is because our method can use the information on observation labels. For type (B), while both our method and the baseline achieved around 1 for ARI, the Acc of ours exceeded that of the baseline by more than 10%. This is because ours can distinguish states by observation values as well as action sequences. In addition, we can see the importance of increasing the number of equivalence test samples by comparing the case of 50 and 200. For type (C), the ARI(6) and ARI(8) of ours exceeded those of the baseline by around 2%. This difference came from whether the paths including rollback were correctly distinguished. Since the baseline does not use the information on observations, it was inclined to incorrectly identify state 4 with state 5. Such misestimation appears, especially in the case in which an action sequence is long such as n = 6 or 8. As a whole, we see that compared to the baseline, our method achieves higher accuracy in OLDFA construction and makes the required number of sampling smaller by nearly half; hence, the required time of construction is shorter. Considering that the baseline requires at most a polynomial number of actions with respect to the type of DFA size [7], we can expect that our method also has a good sample complexity.

To investigate the impact of difficulty of clustering on the performance of our method, we conducted additional simulations for type (B). We changed the standard deviation of a Gaussian distribution corresponding to each label from 5 to 80. Figure 5 shows the results averaged over 100 simulations for each setting. The red line denotes the Acc of our method with its standard error, while the blue line shows the degree of difficulty of clustering. Specifically, we chose 6,000 samples (=1,000 samples/label × 6 labels) from the dataset, applied a clustering to them after dimension reduction, and measured the ARI regardless of automaton construction. Thus, the smaller the blue plot is, the more difficult the clustering is. The numbers written closely to the blue plots are the estimated number of clusters. The green broken line and region show the Acc of the baseline and its standard error, respectively. As can be seen from this figure, when the clustering is easy and the number of clusters is correctly estimated (i.e., 6), the Acc of our method greatly exceeds that of the baseline. On the other hand, as the clustering does not work well, the Acc worsens and drops to around that of the baseline. A notable point is the decline of the Acc stops even when the standard deviation of the dataset becomes much larger. This is because in such a case, the estimated number of clusters becomes one, meaning that our algorithm do essentially the same as the L^* algorithm. In this sense, we conclude the impact of difficulty of clustering on the performance of our method as follows. That is, the performance of our method is high when the dataset is well classified with clustering, and converges to those of the L^* algorithm as the dataset becomes difficult to separate.

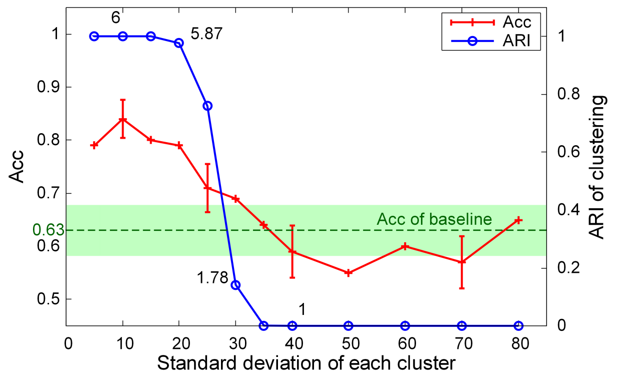


Fig. 5: Impact of difficulty of clustering

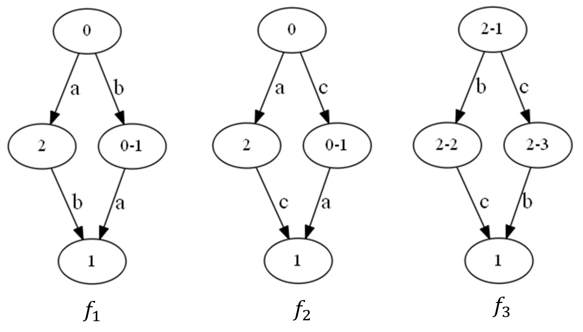


Fig. 6: Constructed OLDFAs

B. Case study: Recovery from container delay

1) Settings: We built a three-tied web architecture system composed of three containers in a Kubernetes 1.16 cluster on a physical node with Ubuntu 18.04. The system consisted of an Nginx container, Rails container, and MySQL container, each of which was deployed in a separated pod. We imposed http requests on this system at random time intervals obeying uniform distribution on [150ms, 200 ms] by using Locust [23], a load test tool. We considered three types of failures, each of which was injected using Pumba [24], a chaos engineering tool: $f_1 = \mathit{Delay}(\text{Nginx}, \text{Rails})$, meaning the Nginx and Rails containers have a delay of $2,000 \pm 100$ ms, similarly $f_2 = \mathit{Delay}(\text{Nginx}, \text{MySQL})$ and $f_3 = \mathit{Delay}(\text{Rails}, \text{MySQL})$. The action space A includes three reboot actions: $a = \texttt{Reboot Nginx}, b = \texttt{Reboot Rails}$, and $c = \texttt{Reboot MySQL}$. We expect that containers with a delay can be recovered by executing the corresponding reboot actions and that irrelevant reboot actions do not change the state. For instance, we expect f_2 to be recovered by the successive actions ca or abc, while not recovered by bbc or ab. The observations are 12-dimensional data consisting of received bytes, transmitted bytes, received packets and transmitted packets at each container. The clustering method was the same as in the previous subsection. The above setting was similar to that in our previous paper [4], in which we addressed automatic recovery with deep reinforcement learning. In this study, we aimed at extracting the background system behavior and visualizing it with interpretable workflow.

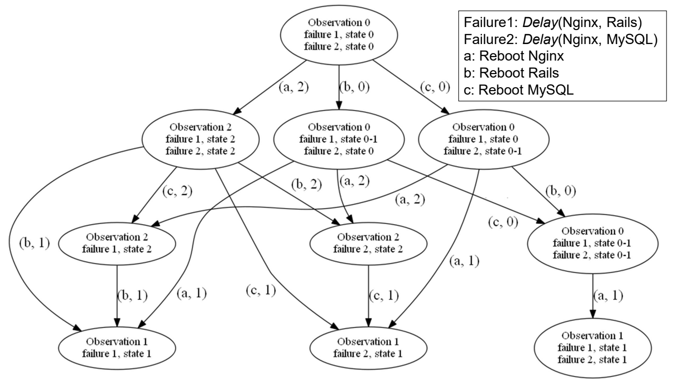

Failure1: Delay(Nginx, Rails)
Failure2: Delay(Nginx, MySQL)
a: Reboot Nginx
b: Reboot Rails
c: Reboot MySQL
Observation 0 failure 1, state 0 failure 2, state 0
(a, 2)
(b, 0)
(c, 0)
Observation 2 failure 1, state 2 failure 2, state 2
Observation 0 failure 1, state 0-1 failure 2, state 0
Observation 0 failure 1, state 0 failure 2, state 0-1
(c, 2)
(b, 2)
(a, 2)
(a, 2)
(b, 0)
(c, 0)
(b, 1)
Observation 2 failure 1, state 2
Observation 2 failure 2, state 2
Observation 0 failure 1, state 0-1 failure 2, state 0-1
(b, 1)
(a, 1)
(c, 1)
(c, 1)
(a, 1)
(a, 1)
Observation 1 failure 1, state 1
Observation 1 failure 2, state 1
Observation 1 failure 1, state 1 failure 2, state 1

Fig. 7: Constructed RPW

2) Results: Figure 6 shows the three OLDFAs constructed with OTS=490. To attach consistent labels through three failures, we ran the algorithm just once with three initial states. Due to the consistency of labels between the three OLDFAs, the label of the initial state for f_1 was not 0. The obtained OLDFAs topologically agreed with our expectation. Note that some distinct states have the same labels. Specifically, we can seen that as far as focusing on observations, Nginx delay dominates other events (label 0), and Rails delay and Nginx delay looks similar (label 2). However, since our method also focuses on action sequences, these states are correctly distinguished.

Since the initial states for f_1 and f_2 cannot be distinguished by observations (label 0), we constructed the RPW from these two OLDFAs. Figure 7 shows the constructed RPW, which exactly agrees with our expectation. For instance, the RPW explicitly shows six f_2-recovery paths, $\{(a,2),(c,1)\}$, $\{(a,2),(b,2),(c,1)\}$, $\{(b,0),(a,2),(c,1)\}$, $\{(b,0),(c,0),(a,1)\}$, $\{(c,0),(a,1)\}$, and $\{(c,0),(b,0),(a,1)\}$, which is consistent with the fact that the recovery from f_2 requires both actions a and c at least. Moreover, the RPW tells us the set of possible failures and states on the basis of historical actions and observations. This would be helpful, for example, for operators to manually handle a failure or understand the system status when the AI, such as a reinforcement learning agent, automatically interacts with the target system.

VII. Conclusion

We addressed the problem of expressing system behavior during the recovery process in a human-interpretable form. We proposed a method of constructing OLDFAs, a new class of automata that have information on observation distribution, on the basis of proactive operation. We also proposed a method of constructing the RPW, which shows the set of possible failures and states during the recovery process even if the type of failure has not yet been narrowed down. We verified the effectiveness and feasibility of our methods through numerical simulations and experiments in a container-based system.

For further study, we need to consider the situation in which the assumption of sparseness and discreteness of actions is broken. Considering the optimization problem of the recovery process on the RPW is another interesting direction of future work. In addition, we need to experimentally verify our methods in larger systems.

References

[1] "The Self-driving network," https://www.juniper.net/us/en/dm/the-self-driving-network/.

[2] "Zero touch network & Service Management (ZSM)," https://www.etsi.org/technologies/zero-touch-network-service-management/.

[3] C. Rosenthal, L. Hochstein, A. Blohowiak, N. Jones, and A. Basiri, *Chaos engineering*. O'Reilly Media, Incorporated, 2020.

[4] H. Ikeuchi, J. Ge, Y. Matsuo, and K. Watanabe, "A framework for automatic failure recovery in ict systems by deep reinforcement learning," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, to be published.

[5] J. Kawasaki, G. Mouri, and Y. Suzuki, "Comparative analysis of network fault classification using machine learning," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–6.

[6] A. Watanabe, K. Ishibashi, T. Toyono, K. Watanabe, T. Kimura, Y. Matsuo, K. Shiomoto, and R. Kawahara, "Workflow extraction for service operation using multiple unstructured trouble tickets," *IEICE Transactions on Information and Systems*, vol. 101, no. 4, pp. 1030–1041, 2018.

[7] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.

[8] B. Bollig, P. Habermehl, C. Kern, and M. Leucker, "Angluin-style learning of nfa." in *IJCAI*, vol. 9, 2009, pp. 1004–1009.

[9] B. Balle and M. Mohri, "Learning weighted automata," in *International Conference on Algebraic Informatics*. Springer, 2015, pp. 1–21.

[10] S. Ayache, R. Eyraud, and N. Goudian, "Explaining black boxes on sequential data using weighted automata," in *International Conference on Grammatical Inference*, 2019, pp. 81–103.

[11] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5247–5256.

[12] T. Okudono, M. Waga, T. Sekiyama, and I. Hasuo, "Weighted automata extraction from recurrent neural networks via regression on state spaces," *arXiv preprint arXiv:1904.02931*, 2019.

[13] G. Weiss, Y. Goldberg, and E. Yahav, "Learning deterministic weighted automata with queries and counterexamples," in *Advances in Neural Information Processing Systems*, 2019, pp. 8560–8571.

[14] E. Aumayr, M. Wang, and A.-M. Bosneag, "Probabilistic knowledge-graph based workflow recommender for network management automation," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–7.

[15] K. R. Joshi, M. A. Hiltunen, W. H. Sanders, and R. D. Schlichting, "Automatic model-driven recovery in distributed systems," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 2005, pp. 25–36.

[16] K. R. Joshi, M. A. Hiltunen, W. H. Sanders, and R. D. Schlichting, "Probabilistic model-driven recovery in distributed systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 913–928, 2010.

[17] H. Ikeuchi, A. Watanabe, T. Kawata, and R. Kawahara, "Root-cause diagnosis using logs generated by user actions," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[18] M. L. Littman, N. Ravi, E. Fenson, and R. Howard, "An instance-based state representation for network repair," in *AAAI*, 2004, pp. 287–292.

[19] M. L. Littman, N. Ravi, E. Fenson, and R. Howard, "Reinforcement learning for autonomic network repair," in *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 2004, pp. 284–285.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[21] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[22] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[23] "Locust," https://locust.io/.

[24] "Pumba," https://github.com/alexei-led/pumba/.