2016 IEEE 55th Conference on Decision and Control (CDC)
ARIA Resort & Casino
December 12-14, 2016, Las Vegas, USA

Event-based Fault Diagnosis for an Unknown Plant

Mohammad Mahdi Karimi, Ali Karimoddini, Alejandro P White, Ira Wendell Bates

Abstract—**This paper presents an active-learning technique for constructing a fault diagnoser for an unknown finite-state Discrete Event System (DES). The proposed algorithm actively asks some basic queries from an oracle through which the algorithm completes a series of observation tables leading to the construction of the diganoser. The resulting diagnoser is a deterministic-finite-state automaton, which detects and identifies occurred faults by monitoring the observable behaviors of the plant. An illustrative example is provided detailing the steps of the proposed algorithm.**

1. INTRODUCTION

With advances in technologies, autonomous systems are being used for many different applications. Nevertheless, the lack of reliability always challenges the deployment of such autonomous systems [1]–[3]. Therefore, in parallel to efforts on increasing the degree of autonomy in newly engineered systems, we have to immensely improve their reliability.

An important step toward improving reliability of autonomous systems is to diagnose fault occurrences in a timely manner to reduce the effects of faults and recover the system before it crashes. Considering faults as abrupt changes in the system, they can be effectively modelled and handled within Discrete Event Systems (DESs) framework [4]. A DES plant can be modeled as a set of discrete states (representing the operation modes of the system), which may change upon the occurrence of events (changes in sensor readings, commands, or other abrupt changes in the system including faults).

In the literature, there are many approaches and tools available for fault diagnosis for discrete event systems such as Petri Nets [5]–[10], Process Algebra [11]–[14], Statecharts [15], [16], and Automata Theory [17]–[23]. In all of these methods, it is assumed that the normal and faulty models of the system are completely known, which may not be applicable to many real situations.

To address the fault diagnosis problem for unknown DES systems, this paper proposes a novel approach to build a DES diagnoser through an active-learning process. Inspired from L^* Algorithm [24], [25], the proposed active-learning algorithm collects the required information about the plant and builds up a deterministic label transition system for fault diagnosis of the plant. Compared to passive-learning techniques, where a rich set of information and a complete set of examples have to be provided for the learner, the proposed active-learning mechanism actively acquires the sufficient required information through a teacher, avoiding redundant information and prior knowledge about system. The teacher is an expert who can answer some basic queries about the system and observed strings. With this acquired information, the algorithm completes a series of observation tables, which eventually conjectures a correct diagnoser. Since it is not possible to place sensors to observe and detect every possible fault, the proposed approach diagnoses faults by monitoring the performance and external observable behavior of the system. The observable behavior of the system is modeled by the natural projection to the observable event set of the system. The resulting diagnoser then can be used for online health monitoring of the system to evaluate its normal or faulty status. An illustrative example is provided to explain the details of the developed method.

The rest of the paper is organized as follows. Section 2 formulates the diagnosis problem and provides basic notations and definitions. Section 3 details the structure of the proposed diagnoser and develops an active-learning algorithm for constructing the diagnoser. In Section 4, an illustrative example is provided to explain different steps of the proposed algorithm. Finally, Section 5 concludes the paper.

2. Background and Preliminaries

Consider the plant modeled by the automaton G as follows:

$$G = (X, \Sigma, \delta, x_0) \tag{1}$$

where X is the state space, Σ is the event set, $\delta : X \times \Sigma \to 2^X$ is the transition relation and x_0 is the initial state.

Example 1. *Consider an unmanned aerial vehicle (UAV) involved in a search mission to find a particular target. A simple model for this search mission is the automaton $G = (X, \Sigma, \delta, x_0)$, which is shown by a directed graph in Fig. 1. In this model, the event set $\Sigma = \{a, b, f_1, f_2\}$ is partitioned into two subsets: observable events $\Sigma_o = \{a, b\}$ and unobservable event set $\Sigma_{uo} = \Sigma_F = \{f_1, f_2\}$, where the event a is for "searching for a target," the event b is for "traveling back to the hangar," f_1 is a fault event that is*

This work is supported by Air Force Research Laboratory and Office of the Secretary of Defense under agreement number FA8750-15-2-0116, and US ARMY Research Office under agreement number W911NF-16-1-0489.
The authors are with the Department of Electrical and Computer Engineering, North Carolina Agricultural and Technical State University, Greensboro, NC 27411 USA.
Corresponding author: A. Karimoddini. Address: 1601 East Market Street, Department of Electrical and Computer Engineering North Carolina A&T State University Greensboro, NC, US 27411. Email: akarimod@ncat.edu *(Tel: +13362853847).*

activated in the case of "loss of the communication link" and the fault event f_2 is for "fuel leakage". In the case that the UAV loses the communication link, it continues searching around (to possibly get connected again), and if there is a fuel leakage, it quickly returns to the hangar. The events cause transition from one state to another one over the state space $X = \{1,2,3,4,5\}$. The initial state is $x_0 = 1$ and the transitions and their corresponding events are shown by labeled arrows in Figure 1.

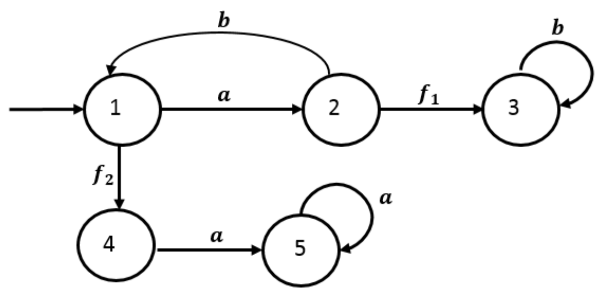


Figure 1. DES model of a UAV involved in a search mission, in which the events a and b are for "searching for a target" and "traveling back to the hangar." The fault events f_1 and f_2 are for "loss of the communication link" and "fuel leakage."

In a DES plant G, the sequence of events forms a *string*, and a set of strings forms a *language*. The string ε denotes the zero-length string, and Σ^* includes all possible strings that can be defined over Σ. The concatenation of the strings s_1 and s_2 is shown by $s_1.s_2$. To mathematically define the language of a system (its all possibly generated strings), we need to revise the definition of δ, which is originally defined over the system's events, and extend it to strings as $\delta(x, s.e) = \delta(\delta(x,s),e)$ for any $s \in \Sigma^*$ and $e \in \Sigma$, and $\delta(x,\varepsilon) = x$. The language of a plant G can then be defined as $\mathcal{L}(G) = \{s \in \Sigma^* | \ \delta(x_0, s) \ \textit{is defined}\}$. Extension-closure of the language $\mathcal{L}$ is denoted by $ext(\mathcal{L})$, where $ext(\mathcal{L}) := \{s \in \Sigma^* | \exists s_p \in \mathcal{L} \ \textit{such that} \ s_p \ \textit{is a prefix of} \ s\}$.

Consider that in plant G, faults $f_1, f_2, \dots,$ and f_n can occur. We assume that these events are modeled as unobservable events in the automaton G, i.e. $\Sigma_F = \{f_1, f_2, \dots, f_n\} \subseteq \Sigma_{uo}$; Otherwise, if faults are observable events, they can be trivially and immediately diagnosed. The observable behavior of the system can be modeled by the natural projection of the language of the system, $\mathcal{L}(G)$, into observable event set, Σ_o, which can be defined as:

- $P(\varepsilon) = \varepsilon$,
- $P(e) = e$, if $e \in \Sigma_o$,
- $P(e) = \varepsilon$, if $e \notin \Sigma_o$,
- $P(s.e) = P(s)P(e)$, for $s \in \Sigma^*$ and $e \in \Sigma$.

Extending the natural projection operator to the language of the plant G as $P(\mathcal{L}(G)) := \{P(s) \mid \forall s \in \mathcal{L}(G)\}$, it is possible to capture the observable behaviors of the system. The inverse projection of a string $w \in \Sigma_o^*$ into $\mathcal{L}(G) \subseteq \Sigma^*$ is $P^{-1}(w) = \{s \in \mathcal{L}(G) \mid P(s) = w\}$, and the inverse projection of a language $\mathcal{L}$ into $\mathcal{L}(G)$ is $P^{-1}(\mathcal{L}) = \bigcup_{w \in \mathcal{L}} P^{-1}(w)$. The diagnosis challenge is then to diagnose faults from the observable behavior of the system which can be modelled as $P(\mathcal{L}(G))$.

Example 2. *In plant G in Example 1, imagine we have observed the string ab. This string in fact could be the projection of the strings $s_1 = af_1b$, $s_2 = abf_2$, or $s_3 = ab$ to the observable event set Σ_o as $P(af_1b) = P(abf_2) = P(ab) = ab$. Strings s_1 and s_2 are faulty with the fault types f_1 and f_2, while s_3 is a normal (non-faulty) string. Therefore, there is an ambiguity if the faults f_1 and f_2 have occurred in the original system. However, if the system continues working and the string aba is observed, then it can be verified that this string can only be the projection of the string abf_2a, concluding that the fault f_2 has occurred. This has to be investigated for all possible strings of the plant G, to solve the diagnosis problem for plant G.*

Inspired by this example, the fault diagnosis problem can now be formally defined as follows:

Problem 1. *In a discrete event system G, for any generated string $s \in \mathcal{L}(G)$, from the observable part, $P(s)$, determine if a fault has occurred; if yes, diagnose the type of the occurred fault.*

3. CONSTRUCTING THE DIAGNOSER

Studying an unknown plant G, we aim to diagnose occurred faults in G by addressing Problem 1. For this purpose, we develop a diagnosis tool, so called diagnoser, whose general structure is shown in Fig. 2. The diagnoser G_d can be described by a tuple:

$$G_d = (Q_d, \Sigma_d, \Delta, \delta_d, h, q_0) \tag{2}$$

where Q_d is the set of diagnoser states, $\Sigma_d = \Sigma_o$ is the event set, δ_d is the transition rule, Δ is the output label set, $h : Q_d \to 2^{\Delta}$ is the output function and q_0 is the initial state. The output label set, Δ, is as:

$$\Delta = \{N\} \cup \{L_1, L_2, \dots, L_m\}, L_i \in \{F_i, A_i\} \tag{3}$$

where N, F_i, and A_i stand for "normal," "occurrence of the fault f_i" and "ambiguity in the occurrence of the fault f_i," respectively.

The proposed algorithm constructs the diagnoser G_d by asking two types of basic queries from an oracle:

- Membership queries: in which the algorithm asks whether a string s belongs to $P(\mathcal{L}(G))$, and if it is faulty.
- Equivalence queries: in which the algorithm asks whether $\mathcal{L}(G_d) = P(\mathcal{L}(G))$. If not, the oracle returns a counterexample $cex \in \mathcal{L}(G_d) \backslash P(\mathcal{L}(G)) \cup P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d)$.

This information will be sorted in series of observation tables. Each observation table is a 3-tuple (S, E, T), where $S \subseteq \Sigma^*$ is a non-empty, prefix-closed, finite set of strings; E is a non-empty, suffix-closed, finite set of strings, and $T(s) : (S \cup S.\Sigma_o).\ E \to 2^{\Delta \cup \{0\}}$ is the condition map. The

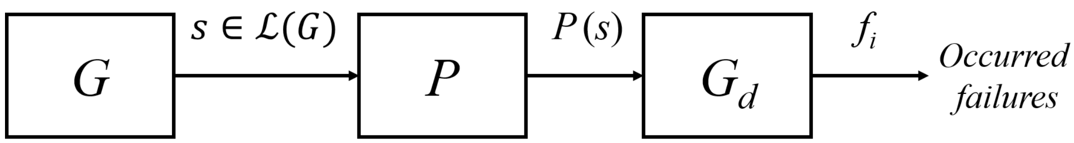


Figure 2. Fault Diagnoser structure

observation table T is a 2-dimensional array, whose rows and columns are labeled by strings $s \in (S \cup S.\Sigma_o)$ and $t \in E$, respectively. The entries of the tables are determined by the condition map, T. For any $w = s.t$ with $s \in (S \cup S.\Sigma_o)$ and $t \in E$, $T(w)$ is determined as follows:

- $T(w) = \{0\}$ if $w \notin P(\mathcal{L}(G))$ (i.e., w is not in the observable part of the system's language).
- $T(w) = \{N\}$ if $w \in P(\mathcal{L}(G))$, and for any $u \in P^{-1}(w)$, $f_i \notin u$, for all $i = 1, \ldots, n$ (i.e., w is a normal (non-faulty) observation).
- $T(w) = \{L_1, L_2, \ldots, L_m\}, L_i \in \{F_i, A_i\}$ where:
  - $F_i \in T(w)$ if all $u \in P^{-1}(w)$ contains the fault f_i (i.e., w is the observation of a faulty string of type f_i).
  - $A_i \in T(w)$ if the observation of w creates ambiguity in occurrence of fault f_i, meaning that $\exists u, u' \in P^{-1}(w)$ such that $f_i \in u$ and $f_i \notin u'$.

To enhance the algorithm we introduce a label propagation mechanism, which automatically extracts information from the observation tables and answers some of the queries without referring to the teacher. The label propagation mechanism can be explained as follows:

1) The fault labels are propagated to keep track of the occurrence of the faults in the past. Hence, if a string s is faulty, so are all its possible extensions:

$$[s \in S \cup S.\Sigma : F_i \in T(s)] \Rightarrow [\forall s' \in ext(s) \cap P(\mathcal{L}(G)) : F_i \in T(s')]. \quad (4)$$

2) For any string s that is not defined in the system, so are all its extensions:

$$[s \in S \cup S.\Sigma : T(s) = \{0\}] \Rightarrow [\forall s' \in ext(s) : T(s') = \{0\}]. \quad (5)$$

We start with the first observation table, T_1, in which $S = E = \{\varepsilon\}$, and then, we will fill up the table by applying the function $T(s) : (S \cup S.\Sigma_o).E \to 2^{\Delta \cup \{0\}}$. Each row in the table can be shown by a function $row : (S \cup S.\Sigma_o).E \to (2^{\Delta \cup \{0\}})^{|E|}$.

Example 3. *In Figure 3(a), the first observation table, T_1, is constructed for the automaton G in Example 1, in which $S = E = \{\varepsilon\}$, and $S.\Sigma_o = \{a, b\}$. For the strings $s = \varepsilon$, $s = a$, and $s = b$, we have respectively $P^{-1}(\varepsilon) = \{\varepsilon, f_2\}$, $P^{-1}(a) = \{a, af_1, f_2a\}$ and $P^{-1}(b) = \varnothing$. Therefore, we have $T(b.\varepsilon) = \{0\}$ as $b \notin P(\mathcal{L}(G))$. Also, we have $T(a.\varepsilon) = \{A_1 A_2\}$ as there exists ambiguity in the occurrence of the faults f_1 and f_2 (it cannot be determined if the string a is observed due to the execution of a, af_1, or f_2a in the plant). Similarly, it can be verified that $T(\varepsilon.\varepsilon) = \{A_2\}$. These values of T have been used to fill up table T_1.*

Definition 1. *An observation table is said to be closed if and only if:*

$$\forall t \in S.\Sigma_o, \exists s \in S \ such \ that \ row(s) = row(t) \quad (6)$$

If an observation table is not closed, it means that there exists a string $t \in S.\Sigma_o$ such that $row(t)$ is different from $row(s)$ for all $s \in S$. To make the observation table closed, it is sufficient to add the string t to S, and extend T to the new table, accordingly.

Example 4. *The observation table T_1 in Figure 3(a) is not closed as neither of the rows in S are equal to $row(b)$ and $row(a)$, $a, b \in (S.\Sigma - S)$. Therefore, to make it closed, a and b are added to S. The updated table is called T_2 and is shown in Figure 3(b).*

Definition 2. *An observation table is said to be consistent if and only if:*

$$\forall s_1, s_2 \in S \ with \ [row(s_1) = row(s_2)] \Rightarrow [row(s_1.\sigma) = row(s_2.\sigma)], \forall \sigma \in \Sigma_o \quad (7)$$

If an observation table is not consistent, there exist two strings $s_1, s_2 \in S$ with $row(s_1) = row(s_2)$, $\sigma \in \Sigma_o$ and $e \in E$, such that $T(s_1\sigma.e) \neq T(s_2\sigma.e)$. Therefore, to make the observation table consistent, it is sufficient to add $\sigma.e$ to E, and extend T to the new table, accordingly.

Example 5. *The observation table T_5 in Figure 3(f) is not consistent. This can be simply verified by letting $s_1 = a$ and $s_2 = ab$, $s_1, s_2 \in S$, for which we have $row(s_1) = row(s_2) = \{A_1 A_2\}$, whereas for $b \in \Sigma_o$, $row(s_1.b) = \{A_1 A_2\} \neq row(s_2.b) = \{F_1\}$. To make the table consistent, the event b is added to E. The updated table is called T_6 and is shown in Figure 3(g).*

For a complete (closed and consistent) observation table, we can construct the diagnoser $G_d(T_i) =$

T_1		E: ϵ
S	ϵ	A_2
$S\Sigma_0 - S$	a	A_1A_2
	b	0

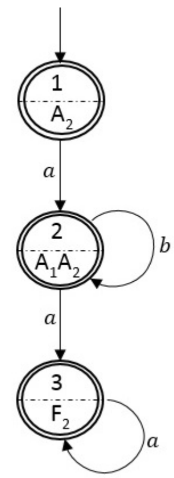
(a)

T_2		E: ϵ
S	ϵ	A_2
	a	A_1A_2
	b	0
$S\Sigma_0 - S$	aa	F_2
	ab	A_1A_2
	ba	0
	bb	0

(b)

T_3		E: ϵ
S	ϵ	A_2
	a	A_1A_2
	b	0
	aa	F_2
$S\Sigma_0 - S$	ab	A_1A_2
	ba	0
	bb	0
	aaa	F_2
	aab	0

(c)



(d)

T_4		E: ϵ
S	ϵ	A_2
	a	A_1A_2
	b	0
	aa	F_2
	ab	A_1A_2
	aba	A_1A_2
	$abab$	A_1A_2
$S\Sigma_0 - S$	ba	0
	bb	0
	aaa	F_2
	aab	0
	abb	F_1
	$abaa$	F_2
	$ababa$	A_1A_2
	$ababb$	F_1

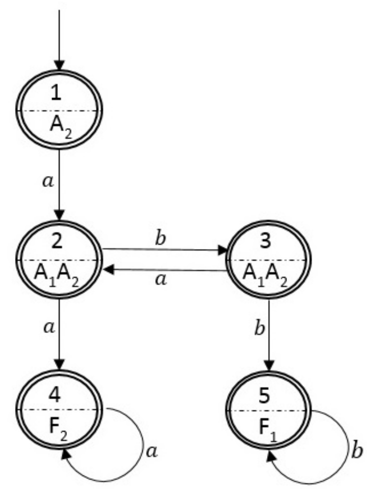
(e)

T_5		E: ϵ
S	ϵ	A_2
	a	A_1A_2
	b	0
	aa	F_2
	ab	A_1A_2
	aba	A_1A_2
	abb	F_1
	$abab$	A_1A_2
$S\Sigma_0 - S$	ba	0
	bb	0
	aaa	F_2
	aab	0
	$abaa$	F_2
	$ababa$	A_1A_2
	$ababb$	F_1
	$abba$	0
	$abbb$	F_1

(f)

T_6		E: ϵ	E: b
S	ϵ	A_2	0
	a	A_1A_2	A_1A_2
	b	0	0
	aa	F_2	0
	ab	A_1A_2	F_1
	aba	A_1A_2	A_1A_2
	abb	F_1	F_1
	$abab$	A_1A_2	F_1
$S\Sigma_0 - S$	ba	0	0
	bb	0	0
	aaa	F_2	0
	aab	0	0
	$abaa$	F_2	0
	$ababa$	A_1A_2	A_1A_2
	$ababb$	F_1	F_1
	$abbb$	F_1	F_1
	$abba$	0	0

(g)



(h)

Figure 3. Constructing the observation tables and the diagnoser for the DES plant in Example 1: **(a)** The observation table T_1 with $S = E = \varnothing$; **(b)** The observation table T_1 is not closed, so a and b are added to S to form T_2; **(c)** The observation table T_2 is not closed, so aa is added to S to form T_3; **(d)** The conjectured automaton $G_d(T_3)$ for the complete observation table T_3; **(e)** The counterexample $cex = abab$ is returned by the oracle for T_3, and hence, the counter example and all its prefixes are added to S to form T_4; **(f)** The observation table T_4 is not closed, so abb is added to S to form T_5; **(g)** The observation table T_5 is not consistent as $row(a) = row(ab)$, but $row(a.b) \neq row(ab.b)$, and hence, b is added to E to form T_6; **(h)** The conjectured automaton for the complete table T_6, which is the final diagnoser for the DES plant in Example 1. The membership queries to the oracle are shown in bold red.

$CoAc(Q_d, \Sigma_d, \Delta_d, \delta_d, h, Q_m, q_0)$ as follows:

$$\begin{gathered} Q_d = \{row(s) | s \in S\} \\ \Sigma_d = \Sigma_o \\ \Delta_d = \Delta \cup \{0\} \\ \delta_d(row(s), \sigma) = row(s.\sigma) \\ h(row(s)) = T(s.\varepsilon) \\ Q_m = \{row(s) | s \in S \text{ and } h(row(s)) \neq 0\} \\ q_0 = row(\varepsilon) \end{gathered} \tag{8}$$

where $CoAc$ is an operator that removes the states from which there does not exist a path to marked states, and Q_m is the set of marked (desired) states.

Example 6. *Figure 3(d) shows the automaton $G_d(T_3)$, constructed for the observation table T_3 in Figure 3(c). $G_d(T_3)$ has three states which are labeled by 1, 2, and 3 at the upper part of the circles, representing the distinct, non-zero rows in S: $row(\epsilon)$, $row(a)$, and $row(aa)$, respectively. Due to the construction procedure, the operator CoAc removes the non-*

marked states which correspond to zero-rows in the table and only leaves the marked states ($Q_d = Q_m$). The output function values for each state are shown at the lower part of the circles. The initial state is q_0, which corresponds to $row(\epsilon)$, and is shown by an entering arrow. The transition rule, δ_d, is shown by labeled, directed arrows connecting the states of the constructed automaton.

Algorithm 1 Learning diagnoser algorithm

```
1:  input: The observable event set, Σ_o, and the observable language of the system P(L(G))
2:  output: The diagnoser G_d with L(G_d) = P(L(G)) which is consistent with T
3:  Initialization: Set i = 1, S = E = {ε}, and form S.Σ, accordingly.
4:  Use the membership queries to build the observation table T_1(S, E, T).
5:  while T_i(S, E, T) is not complete do
6:      if T_i is not closed then
7:          Find s_1 ∈ S and σ ∈ Σ_o such that row(s_1.σ) is different from row(s) for all s ∈ S;
8:          Add s_1.σ to S;
9:          Set i = i + 1;
10:         Update T_i for (S ∪ S.Σ_o).E using the label propagation mechanism and membership queries;
11:     end if
12:     if T_i is not consistent then
13:         Find s_1, s_2 ∈ S, σ ∈ Σ_o and e ∈ E such that row(s_1) = row(s_2) but T(s_1.σ.e) ≠ T(s_2.σ.e);
14:         Add σ.e to E;
15:         Set i = i + 1;
16:         Update T_i for (S ∪ S.Σ).E using the label propagation mechanism and membership queries;
17:     end if
18: end while
19: Construct the automaton G_d(T_i) using (8).
20: Ask equivalence query.
21: if The teacher replies with the counterexample cex then
22:     Add cex and its prefixes to S;
23:     Set i = i + 1;
24:     Update T_i for (S ∪ S.Σ).E using the label propagation mechanism and membership queries;
25:     Go to line 5.
26: end if
27: return: G_d(T_i)
```

After constructing the diagnoser automaton, $G_d(T_i)$, the diagnoser keeps running until a counterexample, $cex \in \mathcal{L}(G_d(T_i)) \backslash P(\mathcal{L}(G)) \cup P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d(T_i))$, is detected by the teacher. In this case, the counterexample cex and all its prefixes will be added to S, and then, the table will be updated with the new changes. This new table again has to be checked for closeness and consistency with $T(s)$.

Example 7. *Figure 3(d) shows the diagnoser $G_d(T_3)$, which is constructed for the observation table T_3 in Figure 3(c). It can be seen that $\mathcal{L}(G_d(T_3))$ is not equivalent to $P(\mathcal{L}(G))$ as $s = abab \in P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d(T_3))$. Therefore, in response to the equivalence query, the oracle returns $cex = abab$. Correspondingly, the string $cex = abab$ and all its prefixes are added to S. The updated table is called T_4 and is shown in Figure 3(f).*

This procedure, starting from the initialization of the algorithm, making the observation tables closed and consistent, and checking for counterexamples can be continued until the algorithm returns the correct diagnoser. This process of constructing the diagnoser G_d is summarized in Algorithm 1.

4. ILLUSTRATIVE EXAMPLE

To illustrate the procedure detailed in Algorithm 1, we use the automaton G in Example 1, which is a model for a UAV that is involved in a simple search mission. The fault events are assumed to be unobservable; Otherwise, if faults are observable events, then they can be trivially and immediately diagnosed. Assume that we do not know this DES model of the plant, and by using Algorithm 1, we are aiming to construct a diagnoser for this DES plant.

We first initialize the algorithm by constructing the observation table T_1, in which $S = E = \varepsilon$ as shown in Figure 3(a). Then, we will fill up the table (S, E, T) by applying T to $S \cup S.\Sigma_o$. The resulting observation table, is not closed as none of the rows in S are equal to $row(a)$ and $row(b)$, $a, b \in (S.\Sigma - S)$. Therefore, a and b are added to S in T_2 as shown in Figure 3(b). Again, it can be seen that T_2 is not closed as none of the rows in S are equal to $row(aa) = \{F_2\}$ for $aa \in (S.\Sigma - S)$. Therefore, aa is added to S in T_3 as shown in Figure 3(c). The observation table T_3 in Figure 3(c) is a complete table, and hence, we can construct the automaton $G_d(T_3)$ using (8) as shown in Figure 3(d).

For the conjectured automaton $G_d(T_3)$, the teacher responds the equivalence query by returning the counterexample $cex = abab \in P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d(T_1))$. Hence, $cex = abab$ and its prefixes are added to S in T_4 as shown in Figure 3(e). Updating T_4, the resulting observation table is not closed as none of the rows in S are equivalent to $row(abb) = \{F_1\}$. Therefore, the string abb is added to S in T_5 (Figure 3(f)). The new observation table, T_5, is not consistent as $row(a) = row(ab)$, but $row(a.b) = \{A_1 A_2\}$ and $row(ab.b) = \{F_1\}$. Hence, b is added to E in T_6 to make it consistent (Figure 3(g)). The observation table T_6, shown in Figure 3(g), is now both closed and consistent, for which the conjectured automaton is shown in Figure 3(h). For the automaton, $G_d(T_6)$, the equivalence query is replied by "Yes" as $P(\mathcal{L}(G)) = \mathcal{L}(G_d(T_6))$. Therefore, the automaton $G_d(T_6)$ is the diagnoser for the plant G. The membership queries to the oracle are shown in bold red in the observation tables in Figure 3. Overall, the diagnoser $G_d(T_6)$ is constructed by actively raising 21 membership queries and two equivalence queries to the oracle.

The diagnoser $G_d(T_7)$ can now be used as a diagnosis tool by synchronizing the diagnoser with the plant. Imagine the string af_1 occurs in the plant G. The diagnoser $G_d(T_7)$

will observe the observable part, a, and will transit to State 2 with the output label A_1A_2 , as at this stage, the diagnoser is not sure whether the faults f_1 and f_2 have occurred or not. When the plant keeps running and generates the string af_1a, then the diagnoser $G_d(T_7)$ observes the string aa, and will transit to State 4 with the output label F_2, informing that the fault f_2 has occurred. This can be verified for all other strings that can be generated by the plant G.

5. CONCLUSION

In this paper, we introduced a new learning-based algorithm for constructing a diagnoser for DES plants. An active-learning technique was developed to construct the diagnoser which detects and identifies occurred faults by monitoring the observable behavior of the plant. The algorithm actively makes two types of queries to a teacher: "the membership queries" and "the equivalence queries". Receiving the answers to these queries, the algorithm gradually completes a series of observation tables leading to the construction of the diganoser. The proposed algorithm was applied to a DES model of a UAV involved in a search mission with multiple types of faults. The corresponding diagnoser was constructed through the proposed active learning mechanism.

Acknowledgment

This research is supported by Air Force Research Laboratory and Office of the Secretary of Defense under agreement number FA8750-15-2-0116 as well as US ARMY Research Office under agreement number W911NF-16-1-0489. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory, ARMY Research Office, OSD, or the U.S. Government.

References

[1] A. Finn and S. Scheding, *Developments and challenges for autonomous unmanned vehicles*. Springer, 2012.

[2] X. Iturbe, K. Benkrid, T. Arslan, I. Martinez, M. Azkarate, and M. D. Santambrogio, "A roadmap for autonomous fault-tolerant systems," in *IEEE Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2010, pp. 311–321.

[3] J. Carreno, G. Galdorisi, S. Koepenick, and R. Volner, "Autonomous systems: Challenges and opportunities," DTIC Document, Tech. Rep., 2010.

[4] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[5] F. Basile, P. Chiacchio, and G. De Tommasi, "An efficient approach for online diagnosis of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 54, no. 4, pp. 748–759, April 2009.

[6] M. P. Cabasino, A. Giua, and C. Seatzu, "Fault detection for discrete event systems using petri nets with unobservable transitions," *Automatica*, vol. 46, no. 9, pp. 1531 – 1539, 2010.

[7] M. Fanti, A. Mangini, and W. Ukovich, "Fault detection by labeled petri nets and time constraints," in *3rd International IEEE Workshop on Dependable Control of Discrete Systems (DCDS)*, 2011, pp. 168–173.

[8] B. Hrúz and M. Zhou, *Modeling and control of discrete-event dynamic systems: With petri nets and other tools*. Springer, 2007, vol. 59.

[9] M. Ioradache and P. Antsaklis, "Resilience to failures and reconfigurations in the supervision based on place invariants," in *Proceedings of the 2004 American Control Conference*, vol. 5, June 2004, pp. 4477–4482 vol.5.

[10] D. Lefebvre and C. Delherm, "Diagnosis of des with petri net models," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 1, pp. 114–118, 2007.

[11] C. Bernardeschi, A. Fantechi, and L. Simoncini, "Formally verifying fault tolerant system designs," *The Computer Journal*, vol. 43, no. 3, pp. 191–205, 2000.

[12] H. Schepers and J. Hooman, "A trace-based compositional proof theory for fault tolerant distributed systems," *Theoretical Computer Science*, vol. 128, no. 1, pp. 127–157, 1994.

[13] N. Dragoni and M. Gaspari, "An object based algebra for specifying a fault tolerant software architecture," *The Journal of Logic and Algebraic Programming*, vol. 63, no. 2, pp. 271 – 297, 2005, special Issue on Process Algebra and System Architecture.

[14] L. Console, C. Picardi, and M. Ribando, "Diagnosis and diagnosability analysis using process algebra," in *Proceedings of the Eleventh International Workshop on Principles of Diagnosis (DX-00), MX, Morelia, Mexico*, 2000, pp. 25–32.

[15] A. Paoli and S. Lafortune, "Diagnosability analysis of a class of hierarchical state machines," *Discrete Event Dynamic Systems*, vol. 18, no. 3, pp. 385–413, 2008.

[16] A. M. Idghamishi and S. H. Zad, "Fault diagnosis in hierarchical discrete-event systems," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1. IEEE, 2004, pp. 63–68.

[17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1555–1575, 1995.

[18] S. H. Zad, R. H. Kwong, and W. M. Wonham, "Fault diagnosis in discrete-event systems: framework and model reduction," *IEEE Transactions on Automatic Control*, vol. 48, no. 7, pp. 1199–1212, 2003.

[19] S. Jiang and R. Kumar, "Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications," *IEEE Transactions on Automation Science and Engineering*, vol. 3, no. 1, pp. 47–59, 2006.

[20] O. Contant, S. Lafortune, and D. Teneketzis, "Diagnosis of intermittent faults," *Discrete Event Dynamic Systems*, vol. 14, no. 2, pp. 171–202, 2004.

[21] W. Qiu and R. Kumar, "Decentralized failure diagnosis of discrete event systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 36, no. 2, pp. 384–395, 2006.

[22] A. White and A. Karimoddini, "Semi-asynchornous failure diagnosis for disctrete event systems," *To appear in IEEE International Conference on Systems, Man, and Cybernetics (SMC2016)*, 2016.

[23] J. Dai, A. Karimoddini, and H. Lin, "Achieving fault-tolerance and safety of discrete-event systems through learning," in *2016 American Control Conference (ACC)*, 2016, pp. 4835–4840.

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

[25] R. L. Rivest and R. E. Schapire, "Inference of finite automata using homing sequences," *Information and Computation*, vol. 103, no. 2, pp. 299–347, 1993.