A Learning-based Approach to Construct a Diagnosis Tool for Discrete Event Systems

Mohammad Mahdi Karimi, et.al

***Abstract*—This paper proposes a novel, systematic active-learning technique for constructing a fault diagnosis tool for a finite-state Discrete Event System (DES). The developed tool, so called diagnoser, detects and identifies occurred faults by monitoring the observable behavior of the plant. The proposed algorithm utilizes an active-learning mechanism to incrementally complete the information about the system. This is achieved by completing a series of observation tables in a systematic way, leading to the construction of the diagnoser. It is shown that the proposed algorithm terminates in a finite number of iterations and returns a correct conjectured diagnoser. The resulting diagnoser is a deterministic-finite-state automaton and is proven that has a minimum number of states. 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 new engineered systems, we have to immensely improve their reliability. The challenge is that despite all our efforts, the potential of occurrence of failures cannot be practically eliminated in the system. So, a major concern in reliable systems is to timely detect, identify, and handle faults. Faults may unpredictably occur everywhere in the system as a result of malfunctioning of sensors or physical instruments, leading to system performance deviation or break down. It is not possible to place sensors to detect every possible fault. Therefore, proper techniques have to be employed to diagnose faults from monitoring the performance and external behavior of the system.

Different approaches have been developed for fault diagnosis such as mathematical model-based techniques [4], [5], identification approaches [6], Data-driven techniques [7], Fuzzy logic [8], Bayesian networks [9], Fault-trees [10], and Neural networks [11]. From a different perspective, many faults can be naturally considered as abrupt changes (events) in the system, and hence, can be effectively treated within the Discrete Event Systems (DES) framework [12]. Moreover, many systems are inherently discrete, or can be well approximated by an abstracted DES model [13], [14]. Therefore, fault diagnosis of DESs has gained considerable attention in the literature [15].

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). In the event-based fault diagnosis approaches [16]–[20], the aim is to detect the occurrence of *faulty events* and identify the nature and type of faults within a finite number of transitions. Alternatively, state-based approaches diagnose if the system has reached *faulty states* [21]–[23]. The effort in [24] combines the advantages of event-based and state-based approaches. 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. In [25], fault diagnosis of DES was studied for a special case of partially unknown systems where a submodel of the system is known.

This paper, however, proposes a novel learning-based approach to build a diagnoser for a DES plant which is initially unknown. Inspired from L^* Algorithm [26], [27], the proposed active-learning algorithm incrementally completes the 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. We prove that the algorithm terminates with returning a correct diagnoser after a finite number of iterations. It is also shown that the resulting DES diagnoser has a minimum number of states. 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. In Section 3, the general structure of the proposed diagnoser is described. Section 4 details the developed active-learning algorithm for constructing the

The authors are with the Department of Electrical and Computer Engineering, North Carolina Agricultural and Technical State University, Greensboro, NC 27411 USA.

diagnoser. In Section 5, an illustrative example is provided to explain different steps of the algorithm. In Section 6, the properties of the proposed algorithm are investigated. Finally, Section 7 concludes the paper and discusses the future works.

2. PROBLEM FORMULATION

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. In the graphical representation of an automaton, states are shown as circles, which are connected by labeled arrows representing the transitions between the states, and the initial state is shown by a circle with an entering arrow. (See, e.g., Fig. 2, which shows the DES model of a helicopter involved in a simple search scenario).

The sequence of events forms a *string*, and a set of strings forms a *language*. With the abuse of notation, we use $e \in s$ to say that the event e belongs to the string s, if the event e is one of the events forming the string s. The concatenation of the strings s_1 and s_2 is shown by $s_1.s_2$. Let Σ^* denote all possible finite-length strings over Σ including the zero-length string ε. Then, δ can be extended to $\delta : X \times \Sigma^* \to 2^X$ as follows:

- $\delta(x, \varepsilon) = x$
- $\delta(x, s.e) = \delta(\delta(x, s), e)$ $for\ any\ s \in \Sigma^*\ and\ e \in \Sigma$

The language of G, denoted by $\mathcal{L}(G)$, is the sequence of strings that can be generated by G, which can be described as $\mathcal{L}(G) = \{s \in \Sigma^* | \ \delta(x_0, s)\ is\ defined\}$. Consider the string $s = s_p.u.s_s$, then s_p and s_s are called the prefix and suffix of the string s, respectively. The language $\mathcal{L}$ is said to be prefix/suffix closed if all prefixes/suffixes of all strings in the language are also a member of the language $\mathcal{L}$. Extension-closure of the language $\mathcal{L}$ is denoted by $ext(\mathcal{L})$, where $ext(\mathcal{L}) := \{s \in \Sigma^* | \exists s_p \in \mathcal{L}\ such\ that\ s_p\ is\ a\ prefix\ of\ s\}$.

The event set Σ can be partitioned into disjoint sets as $\Sigma = \Sigma_o \cup \Sigma_{uo}$, where Σ_o is the set of observable events (those events whose occurrence can be detected, e.g. the changes that can be detected by sensors), and Σ_{uo} is the set of unobservable events.

Consider that in the plant G, faults $f_1, f_2, \ldots,$ and f_n can occur. We assume that these events are modelled as unobservable events in the automaton G, i.e. $\Sigma_F = \{f_1, f_2, \ldots, f_n\} \subseteq \Sigma_{uo}$, otherwise, if faults are observable events, then they can be trivially and immediately diagnosed. We also assume that these faults make a distinct change in the system modelled by a transition $x \xrightarrow{f_i} x'$ in G, but they do not bring the system to a halt mode. Therefore, there will be ample time to diagnose faults from the observable behavior of the system. The observable behavior of the system can be modelled by the natural projection of the language of the system, $\mathcal{L}(G)$, into observable event set, Σ_o. The natural projection of an event e to the set Σ_o can be defined as:

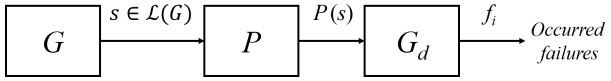

Figure 1. Fault Diagnoser structure

$$P(e) := \begin{cases} e & \text{if } e \in \Sigma_o \\ \varepsilon & \text{otherwise} \end{cases} \tag{2}$$

Extending this definition to strings, we will have $P(\varepsilon) = \varepsilon$ and $P(s.e) := P(s).P(e)$, for any $s \in \Sigma^*$ and $e \in \Sigma$. This definition can be further extended to a language $\mathcal{L}$ as $P(\mathcal{L}) := \{P(s) | \forall s \in \mathcal{L}\}$. Inversely, for the plant G, we define $P^{-1}(s) := \{t \in \mathcal{L}(G) | P(t) = s\}$ and $P^{-1}(\mathcal{L}) := \bigcup_{s \in \mathcal{L}} P^{-1}(s)$.

Using natural projection operator to model the observable behavior of the system, we can now formally describe the fault diagnosis problem 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, and if yes, diagnose the type of the occurred fault.*

3. THE GENERAL STRUCTURE OF THE DIAGNOSER

To address Problem 1, we assume that the system is initially unknown and will develop a learning-based approach to construct a diagnoser, which can be used as a diagnosis tool for the system's faults. The general structure of the proposed diagnoser is shown in Fig. 1.

The diagnoser G_d can be described by a tuple:

$$G_d = (Q_d, \Sigma_d, \Delta, \delta_d, h, q_0) \tag{3}$$

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:

$$\Delta = \{N\} \cup \{L_1, L_2, \ldots, L_m\}, L_i \in \{F_i, A_i\} \tag{4}$$

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 diagnoser takes the observable part of the strings generated by the plant G and performs the diagnosing process in parallel with the system giving out the output labels, which describes the current failure status of the plant.

4. CONSTRUCTING THE DIAGNOSER

The diagnosis problem is to detect and identify the faults from the observable behaviors of the system, $P(\mathcal{L}(G))$. Inspired by the L^* Algorithm [26], we introduce a diagnosis algorithm that can effectively and actively learn the

diagnoser G_d introduced in (3). The algorithm constructs the diagnoser G_d by asking queries from a teacher who can answer two types of basic queries:

- 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 teacher returns a counterexample $cex \in \mathcal{L}(G_d) \backslash P(\mathcal{L}(G)) \cup P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d)$.

With these queries, each new observation will be classified as a faulty/non-faulty member or non-member of $P(\mathcal{L}(G))$. Then, this information will be used to create a series of observation tables which incrementally record and maintain the information about the observed strings. 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 $T(s) : (S \cup S.\Sigma_o).E \rightarrow 2^{\Delta \cup \{0\}}$ is the condition map. This information will be sorted in series of observation tables. The 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, carrying out the membership queries. 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 is in observable part of the system's language.
- $T(w) = \{N\}$ if 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 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 $\exists u, u' \in P_{\Sigma_o}^{-1}(w)$ such that $f_i \in u$ and $f_i \notin u'$.

To reduce the number of queries to the teacher, we use a label propagation mechanism, which can handle some of the above queries using current information in the table. The label propagation mechanism works 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:

$$[s \in S \cup S.\Sigma : F_i \in T(s)] \Rightarrow [\forall s' \in ext(s) \cap P(\mathcal{L}(G)) : F_i \in T(s')]. \quad (5)$$

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

$$[s \in S \cup S.\Sigma : T(s) = \{0\}] \Rightarrow [\forall s' \in ext(s) : T(s') = \{0\}]. \quad (6)$$

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 $T(s) : (S \cup S.\Sigma_o).E \rightarrow 2^{\Delta \cup \{0\}}$. Each row in the table can be shown by a function $row : (S \cup S.\Sigma_o).E \rightarrow (2^{\Delta \cup \{0\}})^{|E|}$. For instance, in Figure 3.a, for the string $s = b$, $P^{-1}(b) = \{b, b.f_1\}$. Therefore, for the second row and first column of this observation table, we have $T(b.\varepsilon) = \{A_1\}$, which means that there exists ambiguity in the occurrence of the fault F_1. To simplify the demonstration of the table, we remove the $zero-row$ strings $S_0 = \{s \in S \cup S.\Sigma_o |\ T(s.\varepsilon) = \{0\}\}$ from $S \cup S.\Sigma_o$. The reason is that for any $s \in S_0$, $row(s) = row(ext(s)) = (\{0\}, \{0\}, \ldots, \{0\})$, meaning that neither s, nor strings in $ext(s)$ are possible to be observed. For example in Figure 3.b, the string $s = a$ is removed as $T(a.\varepsilon) = \{0\}$.

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

$$\forall t \in S.\Sigma_o - S_0, \exists s \in S \ such \ that \ row(s) = row(t) \quad (7)$$

If the observation table is not closed, it means that there exists a string $t \in S.\Sigma_o - S_0$ 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. For example, the observation table in Figure 3.b is not closed as neither of the rows in S are equal to $row(b)$, $b \in (S.\Sigma - S - S_0)$. Therefore, to make it closed, b is added to S as shown in Figure 3.c.

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

$$\forall s_1, s_2 \in S \ with \ [row(s_1) = row(s_2)] \Rightarrow [row(s_1.\sigma) = row(s_2.\sigma)], \forall \sigma \in \Sigma_o \quad (8)$$

If the 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. For example, the observation table in Figure 5.b is not consistent since $row(b) = row(bb)$, but $row(ba) = \{0\}$ and $row(bba) = \{N\}$. To make the table consistent, the event a is added to E as shown in Figure 5.c. As it will be shown in Theorem 2, when we construct an automaton for a consistent observation table, the resulting automaton will be deterministic, i.e., any event changes the state of the automaton to a unique state.

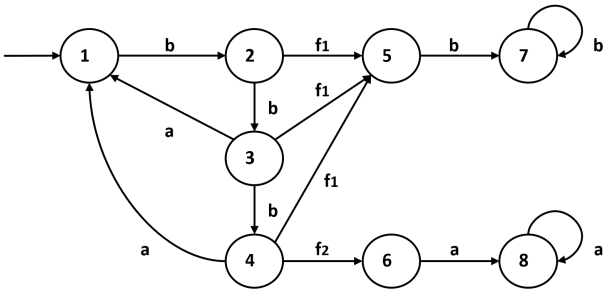

Figure 2. The DES model of a helicopter involved in a simple search mission.

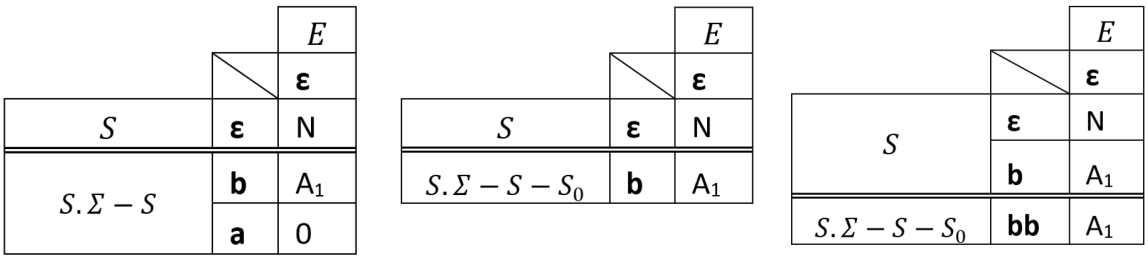

Figure 3. Constructing the observation table T_1. **(a)** Initialized T_1, **(b)** $row(a) = \{0\}$ is removed, **(c)** T_1 was not closed in the previous step, so b is added to S to make it closed.

Algorithm 1 Learning diagnoser algorithm

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

For a complete (closed and consistent) observation table, we can construct the diagnoser $G_d(T_i) = (Q_d, \Sigma_d, \Delta_d, \delta_d, h, q_0)$ as follows:

$$\begin{aligned} Q_d &= \{row(s) | s \in S\} \\ \Sigma_d &= \Sigma_o \\ \Delta_d &= \Delta \cup \{0\} \\ \delta_d(row(s), \sigma) &= row(s.\sigma) \\ h(row(s)) &= T(s.\varepsilon) \\ q_0 &= row(\varepsilon) \end{aligned} \tag{9}$$

After obtaining the conjectured 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 $T(s)$. For example, $G_d(T_1)$ in Figure 4 is not language equivalent to $P(\mathcal{L}(G))$, where G is the DES model of the helicopter shown in Figure 2. The teacher responds the equivalence query by returning the counterexample $cex = bba \in P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d(T_i))$. Therefore, $cex = bba$ and all its prefixes are added to S as shown in Figure 5.a. The whole procedure of constructing the diagnoser G_d is summarized in Algorithm 1.

5. ILLUSTRATIVE EXAMPLE

To illustrate the procedure detailed in Algorithm 1, consider a helicopter that is involved in a simple search mission and is modelled by automaton G, as shown in Fig. 2. In this model, observable events in $\Sigma_o = \{a, b\}$ result in changing the operation mode of the helicopter, where the event a is for "travel back to the hanger", the event b is for "explore". The only unobservable events of this helicopter are fault events $\Sigma_{uo} = \Sigma_F = \{f_1, f_2\}$, where f_1 is for "loss of the communication link" and the event f_2 is for "fuel leakage". In case that the helicopter 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 hanger. Assume that this DES model of the plant is not known to the algorithm. To construct the diagnoser, the algorithm has to 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 $S \cup S.\Sigma_o$. We then remove the zero-row strings (Figure 3.b). The resulting observation table, is not closed as none of the rows in S are equal to $row(b)$, $b \in (S.\Sigma - S - S_0)$. Therefore, b is added to S as shown in Figure 3.c. The observation table T_1 in Figure 3.c is a complete table, and hence, we can construct the automaton $G_d(T_1)$ using (9), which is shown in Figure 4.

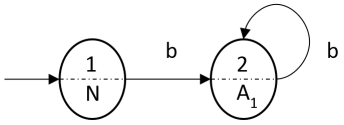

Figure 4. Conjectured automaton $G_d(T_1)$ for the complete observation table T_1 in Figure 3.c.

The teacher responds the equivalence query for $G_d(T_1)$ with the counterexample $cex = bba \in P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d(T_1))$. Hence, $cex = bba$ and its prefixes are added to S as shown in Figure 5.a. The resulting observation table is not closed as none of the rows in S are equivalent to $row(bbb)$. Therefore, the string bbb is added to S (Figure 5.b). The new observation table in Figure 5.b is not consistent as $row(b) = row(bb)$, but $row(ba) = \{0\}$ and $row(bba) = \{N\}$. Hence, a is added to E to make it consistent (Figure 5.c). The resulting observation table is not closed. Theretofore, $bbba$, $bbbb$, and $bbbaa$ are respectively added to the table in Figures 5.d, 5.e, 5.f to make the table closed. The observation table shown in Figure 5.f is now both closed and consistent, for which the conjectured automaton is shown in Figure 6. For this automaton, the equivalence query is replied by "Yes" , and the automaton $G_d(T_2)$ is the diagnoser for the plant G.

The diagnoser $G_d(T_2)$ can be used as a diagnosis tool by synchronizing the diagnoser with the plant. Imagine that the string bf_1 happens in the plant G. The diagnoser $G_d(T_2)$ will observe the observable part, b, and will transit to State 2 with the output label A_1 , as at this stage, the diagnoser is not sure whether the fault f_1 has occurred or not. When the plant keeps running and generates the string bf_1bbb, then the diagnoser $G_d(T_2)$ observes the string $bbbb$, and will transit to State 6 with the output label F_1, informing that the fault f_1 has occurred. This can be verified for all other strings that can be generated by the plant G.

6. PROPERTIES OF THE ALGORITHM

6.1. Minimality and Determinism of the Diagnoser

We first show that the diagnoser G_d can be realized by a finite-state automaton, and then, will prove that the construction algorithm, Algorithm 1, results in a diagnoser with a minimum number of states. Also, we will show that the constructed diagnoser is deterministic.

Lemma 1. *For the finite-state automaton G, its corresponding diagnoser, G_d, can be realized by a finite-state automaton.*

Proof. Since the plant G is a finite state automaton, its language, $\mathcal{L}(G)$, can be expressed by a regular language. For a regular language $\mathcal{L}(G)$, its projection, $P(\mathcal{L}(G))$, can be also expressed by a regular language. We also know that for any regular language, there exists a language equivalent finite-state automaton [28], [29]. Hence, the diagnoser G_d with the regular language $P(\mathcal{L}(G))$ can be realized by a finite-state automaton. □

Theorem 1 (minimality of the Diagnoser)**.** *Let G_d to be the diagnoser constructed by Algorithm 1. Then, any other diagnoser, which is consistent with the condition map, T, has more number of states than G_d.*

Proof. Consider that the diagnoser G_d, constructed by Algorithm 1, has m states. By contradiction, assume that there exists a diagnoser $G'_d = (Q'_d, \Sigma'_d, \Delta', \delta'_d, h', q'_0)$, which is consistent with T and has m' states where $m' < m$. Since G'_d is consistent with T, for any $s \in (S \cup S.\Sigma_o)$ and $e \in E$, we have $h'(\delta'_d(q'_0, s.e)) = T(s.e)$. Now, consider distinct strings $s_1, s_2 \in S$, where $row(s_1) \neq row(s_2)$. Since the observation table of G_d is consistent, for distinct strings s_1 and s_2 there exists an event e, such that $T(s_1.e) \neq T(s_2.e)$. Since $h'(\delta'_d(q'_0, s.e)) = T(s.e)$, this will result in $h'(\delta'_d(q'_0, s_1.e)) \neq h'(\delta'_d(q'_0, s_2.e))$, which essentially requires that $\delta'_d(q'_0, s_1.e) \neq \delta'_d(q'_0, s_2.e)$. Therefore, $\delta'_d(q'_0, s_1.e)$ and $\delta'_d(q'_0, s_2.e)$ are distinct states in G'_d. Since, there are m distinct rows in the observation table of G_d, there will be at least m distinct states in G'_d. This means that $m' \geq m$, which contradicts with the assumption. □

Theorem 2. *The diagnoser G_d, constructed by Algorithm 1, is a deterministic finite-state automaton.*

Proof. To prove that G_d has a finite number of states, recall that according to Lemma 1 the diagnoser for the plant G can be realized by a finite-state automaton. Also, based on Theorem 1, the diagnoser G_d, constructed by Algorithm 1, has minimum number of states. Hence, G_d has a finite number of states.

To prove that G_d is deterministic, by contradiction, assume that the automaton G_d, which is conjectured from a complete observation table (S, E, T), is not deterministic. Then, there exists an event $e \in \Sigma_o$, a string $s \in S$ and its corresponding state $q = row(s) \in Q_d$, such that $\delta_d(q, e) = q_1$ and $\delta_d(q, e) = q_2$ and $q_1 \neq q_2$. According to the construction procedure provided in (9), this means that for $s \in S$, $q_1 = row(s.e)$ is different from $q_2 = row(s.e)$, which contradicts with the consistency of the observation table (S, E, T) and concludes that G_d is deterministic. □

6.2. Termination of the Algorithm

The construction algorithm detailed in Algorithm 1 has two loops: one is for completing the observation table and

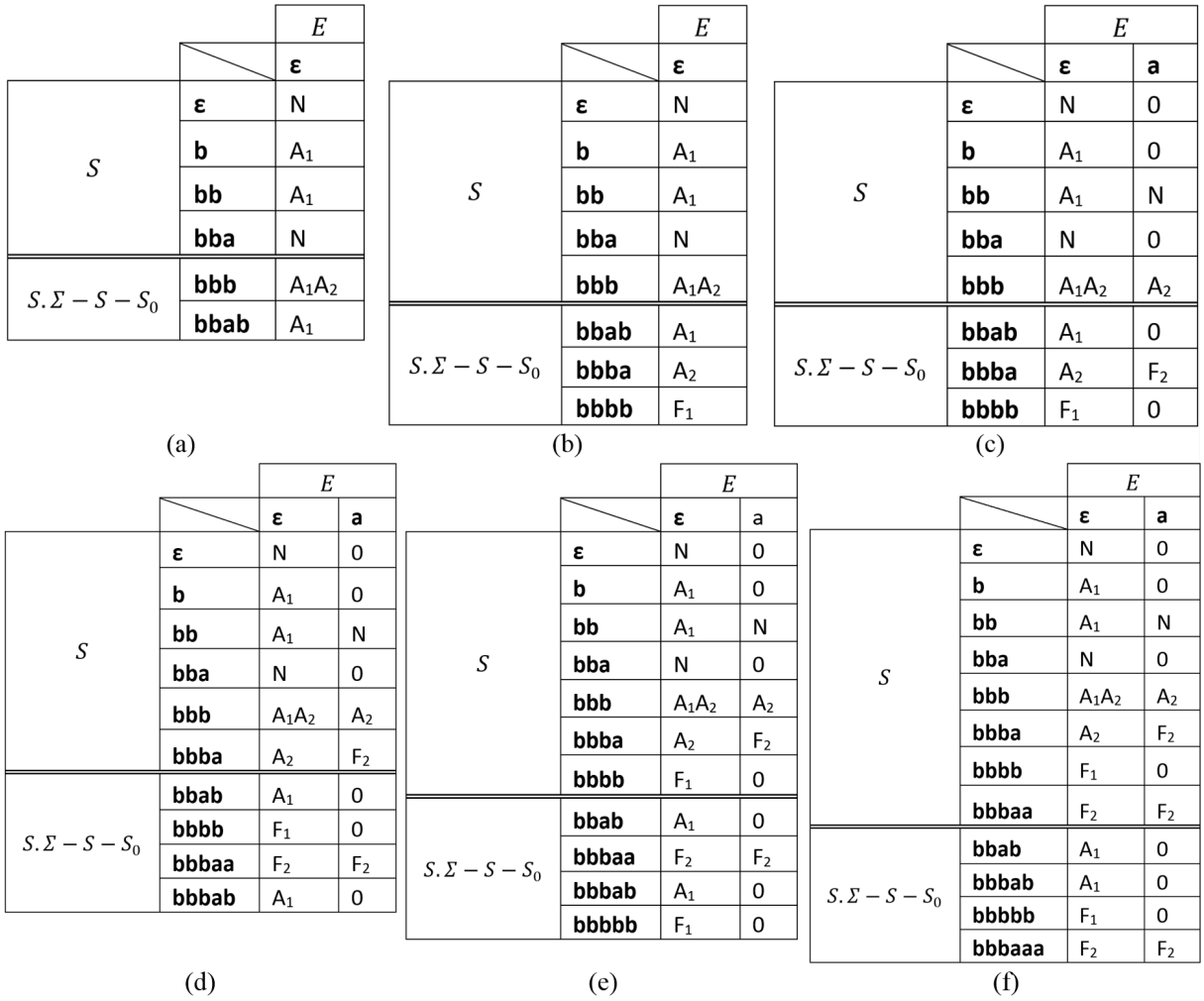

Figure 5. Constructing the observation table T_2. **(a)** The counterexample cex = bba and its prefixes are added to S, **(b)** The observation table in the previous step was not closed, so bbb is added to S, **(c)** The observation table in the previous step was not consistent, so a is added to E to make it consistent, **(d)** The observation table in the previous step was not closed, so $bbba$ is added to S, **(e)** The observation table in the previous step was not closed, so $bbbb$ is added to S, **(f)** The observation table in the previous step was not closed, so $bbbaa$ is added to S.

the other one for classifying the counterexamples returned by the teacher.

Lemma 2. *The observation table for the diagnoser G_d can be always completed within finite number of iterations.*

Proof. If the observation table is not closed, there exists a string $s_1 \in S$ and $\sigma \in E$, such that $row(s_1.\sigma)$ is different from $row(s)$ for all $s \in S$. To make the observation table closed, we should add $s_1.\sigma$ to S, which will create a new distinct row in S. Correspondingly, one state will be added to the states of G_d.

Also, if the observation table is not consistent, there exist strings $s_1, s_2 \in S$, $\sigma \in \Sigma_0$, and $e \in E$ such that $row(s_1) = row(s_2)$, but $T(s_1.\sigma.e) \neq T(s_2.\sigma.e)$. Then, to make the table consistent, we have to add $\sigma.e$ to E to distinguish $row(s_1)$ and $row(s_2)$. Therefore, these rows become distinct in S, meaning that the number of distinct rows in the observation table (S, E, T) will increase at least one. Correspondingly, at least one state will be added to the states of G_d.

Thus, making the observation closed and consistent increases the number of states of G_d, monotonically. According to Theorem 2, the number of states in G_d is finite; hence, the process of making the observation table closed and consistent cannot be continued forever and will be terminated after a finite number of iterations returning a completed observation table. □

Lemma 3. *The number of counterexamples that the teacher in Algorithm 1 returns in response to equivalence queries are finite.*

Proof. If the conjectured automaton $G_d(T_i)$ for the table $T_i(S, E, T)$ is not correct due to a counterexample $cex \in \mathcal{L}(G_d) \backslash P(\mathcal{L}(G)) \cup P(\mathcal{L}(G)) \backslash \mathcal{L}(G_d)$, Algorithm 1 adds cex and all its prefixes to S, and then, the algorithm updates the observation table. If the new table is

not complete, according to Lemma 2, it is always possible to make it complete. Let's call the new completed table $T_{i+1}(S', E', T')$. Using (9), we can conjecture the automaton $G_d(T_{i+1})$ for the complete observation table $T_{i+1}(S', E', T')$. According to Theorem 1, the conjectured automaton $G_d(T_i)$ and $G_d(T_{i+1})$ are with minimum number of states. For these two minimum-state automata, $G_d(T_{i+1})$ can classify the string cex but $G_d(T_i)$ cannot. Hence, $G_d(T_{i+1})$ has at least one state more than $G_d(T_i)$. Therefore, taking the counterexamples into account increases the number of states, monotonically. Based on Theorem 2, the number of states of the diagnoser is finite. Hence, the process of returning a counterexample and extending the observation table to classify the returned counterexamples cannot be continued forever and will be terminated after a finite number of iterations returning a correct conjectured diagnoser G_d. □

Theorem 3. *The algorithm for constructing the diagnoser G_d, Algorithm 1, terminates after a finite number of iterations for completing the observation table and classifying the counterexamples.*

Proof. Algorithm 1 has two loops: one is for completing the observation table and the other one for classifying the counterexamples. Lemma 2 and Lemma 3 guarantee that these two loops terminate in finite number of iterations, so does Algorithm 1. □

7. 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 will actively make 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. It was proved that the algorithm terminates with returning a correct diagnoser automaton. It is also shown that the resulting diagnoser automaton has a minimum number of states and is a deterministic-finite-state automaton. The algorithm can be applied to a plant which is initially unknown and may have multiple faults. Future study directions of this work may include examining the scalability of the proposed approach to extend the proposed framework to a decentralized and distributed diagnosis architectures.

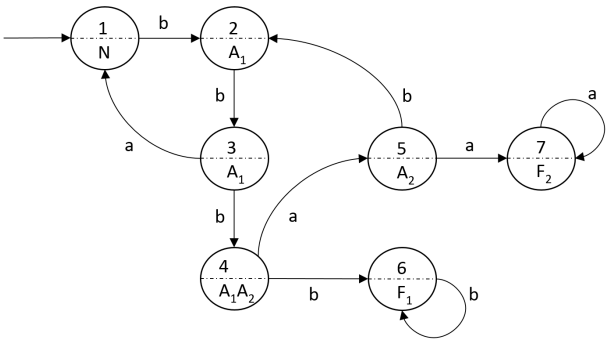

Figure 6. Conjectured automaton $G_d(T_2)$ for the complete observation table T_2 in Figure 5.f.

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] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part i: Quantitative model-based methods," *Computers & chemical engineering*, vol. 27, no. 3, pp. 293–311, 2003.

[5] J. Chen and R. J. Patton, *Robust model-based fault diagnosis for dynamic systems*. Springer Science & Business Media, 2012.

[6] S. Simani, C. Fantuzzi, and R. J. Patton, *Model-based fault diagnosis in dynamic systems using identification techniques*. Springer Science & Business Media, 2013.

[7] E. L. Russell, L. H. Chiang, and R. D. Braatz, *Data-driven methods for fault detection and diagnosis in chemical processes*. Springer Science & Business Media, 2012.

[8] Y. Zheng, H. Fang, and H. O. Wang, "Takagi-sugeno fuzzy-model-based fault detection for networked control systems with markov delays," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 4, pp. 924–929, 2006.

[9] U. Lerner and R. Parr, "Bayesian fault detection and diagnosis in dynamic systems," in *Proc. AAAI*, 2000, pp. 531–537.

[10] W. Lee, D. Grosh, F. Tillman, and C. Lie, "Fault tree analysis, methods, and applications," *IEEE Transactions on Reliability*, vol. R-34, no. 3, pp. 194–203, Aug 1985.

[11] Y. Maki and K. Loparo, "A neural-network approach to fault detection and diagnosis in industrial processes," *IEEE Transactions on Control Systems Technology*, vol. 5, no. 6, pp. 529–541, Nov 1997.

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

[13] J. Raisch and S. O'Young, "Discrete approximation and supervisory control of continuous systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 569–573, Apr 1998.

[14] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, July 2000.

[15] J. Zaytoon and S. Lafortune, "Overview of fault diagnosis methods for discrete event systems," *Annual Reviews in Control*, vol. 37, no. 2, pp. 308–320, 2013.

[16] 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.

[17] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 8, pp. 1318–1321, Aug 2001.

[18] T.-S. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 9, pp. 1491–1495, Sep 2002.

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

[20] 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.

[21] F. Lin, "Diagnosability of discrete event systems and its applications," *Discrete Event Dynamic Systems*, vol. 4, no. 2, pp. 197–212, 1994.

[22] 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.

[23] 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.

[24] M. Sayed-Mouchaweh, A. Philippot, and V. Carre-Menetrier, "Decentralized diagnosis based on boolean discrete event models: application on manufacturing systems," *International journal of production research*, vol. 46, no. 19, pp. 5469–5490, 2008.

[25] R. H. Kwong and D. L. Yonge-Mallo, "Fault diagnosis in discrete-event systems: incomplete models and learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 41, no. 1, pp. 118–130, 2011.

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

[27] 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.

[28] A. Aho and J. Ullman, *Introduction to automata theory, languages and computation*. Addison-Wesley, Reading, MA, 1979.

[29] R. Kumar and V. K. Garg, *Modeling and control of logical discrete event systems*. Springer Science & Business Media, 2012, vol. 300.