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 diagnoser. 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:

$$G = (X, \Sigma, \delta, x_0)$$

where $X$ is the state space, $\Sigma$ is the event set, $\delta : X \times \Sigma \rightarrow 2^X$ 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
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.”](image)

In a DES plant $G$, the sequence of events forms a string, and a set of strings forms a language. The string $\varepsilon$ denotes the zero-length string, and $\Sigma^*$ includes all possible strings that can be defined over $\Sigma$. The concatenation of the strings $s_1$ and $s_2$ is shown by $s_1 \cdot s_2$. To mathematically define the language of a system (its all possible generated strings), we need to revise the definition of $\delta$, 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 $L(G) = \{s \in \Sigma^* | \delta(x_0, s) \text{ is defined}\}$. Extension-closure of the language $L$ is denoted by $\text{ext}(L)$, where $\text{ext}(L) = \{s \in \Sigma^* | \exists s' \in L \text{ such that } s_0 \text{ is a prefix of } s\}$.

Consider that in plant $G$, faults $f_1, f_2, \ldots,$ and $f_n$ can occur. We assume that these events are modeled as unobservable events in the automaton $G$, i.e. $\Sigma_F = \{f_1, f_2, \ldots, f_n\} \subseteq \Sigma_0$; 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, $L(G)$, into a observable event set, $\Sigma_o$, which can be defined as:

- $P(\varepsilon) = \varepsilon$,
- $P(e) = e$, if $e \in \Sigma_o$,
- $P(\varepsilon) = \varepsilon$, 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 $P(L(G)) := \{P(s) | \forall s \in L(G)\}$, it is possible to capture the observable behaviors of the system. The inverse projection of a string $w \in \Sigma^*_o$ into $L(G) \subseteq \Sigma^*$ is $P^{-1}(w) = \{s \in L(G) | P(s) = w\}$, and the inverse projection of a language $L$ into $L(G)$ is $P^{-1}(L) = \bigcup_{w \in L} P^{-1}(w)$.

The diagnosis challenge is then to diagnose faults from the observable behavior of the system which can be modeled as $P(L(G))$.

**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 $s_1 = a_f b$, $s_2 = abf_2$, or $s_3 = ab$ to the observable event set $\Sigma_o$ as $P(a_f b) = P(abf_2) = P(ab) = ab$. 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 $s \in L(G)$, 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:

$$G_d = (Q_d, \Sigma_d, \Delta, \delta_d, h, q_0)$$

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

$$\Delta = \{N \cup \{L_1, L_2, \ldots, L_m\}, L_i \in \{F_1, A_1\}\}$$

where $N$, $F_1$, and $A_1$ stand for “normal,” “occurrence of the fault $f_1$,” and “ambiguity in the occurrence of the fault $f_1$,” 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 $P(L(G))$, and if it is faulty.
- Equivalence queries: in which the algorithm asks whether $L(G_d) = P(L(G))$. If not, the oracle returns a counterexample $cex \in L(G_d) \backslash P(L(G)) \cup P(L(G)) \backslash L(G_d)$.

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 $T(s) : (S \cup S.\Sigma_o), E \rightarrow 2^{\Delta \cup \{0\}}$ is the condition map. The
observation table $T$ is a 2-dimensional array, whose rows and columns are labeled by strings $s \in (S \cup S_o) = T(t)$. The entries of the tables are determined by the condition map, $T$. For any $w = s.t$ with $s \in (S \cup S_o)$ and $t \in E$, $T(w)$ is determined as follows:

- $T(w) = \{0\}$ if $w \notin P(L(G))$ (i.e., $w$ is not in the observable part of the system’s language).
- $T(w) = \{N\}$ if $w \notin P(L(G))$, 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).
- $T(w) = \{L_1, L_2, \ldots, L_m\}$, $L_i \in \{\text{fault} \} \text{ where:}$
  - $F_i \in T(w)$ 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:

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

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

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

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 $T(s) : (S \cup S_o) \rightarrow E$. Each row in the table can be shown by a function $row : (S \cup S_o) \rightarrow (2^{\Delta_o(0)})^E$.

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 $S_o = \{a, b\}$. For the strings $s = \varepsilon$, $s = a$, and $s = b$, we have respectively $P^{-1}(\varepsilon) = \{a, f_2\}$, $P^{-1}(a) = \{a, af_1, f_2a\}$ and $P^{-1}(b) = \varnothing$. Therefore, we have $T(b, \varepsilon) = \{0\}$ as $b \notin P(L(G))$. 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 $a$, a $f_1$, 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:

$$\forall t \in S \cup S_o, \exists s \in S \text{ such that } row(s) = row(t) \quad (6)$$

If an observation table is not closed, it means that there exists a string $t \in S \cup S_o$ 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 $row(a)$, $a, b \in (S \cup S_o - S)$. 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:

$$\forall s_1, s_2 \in S \text{ with } row(s_1) = row(s_2) \Rightarrow row(s_1, \sigma) = row(s_2, \sigma), \forall \sigma \in \Sigma_o \quad (7)$$

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 $\varepsilon \in E$, such that $T(s_1, \sigma, \varepsilon) \neq T(s_2, \sigma, \varepsilon)$. Therefore, to make the observation table consistent, it is sufficient to add $\sigma, \varepsilon$ 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).
Figure 3. Constructing the observation tables and the diagnoser for the DES plant in Example 1: (a) The observation table $T_1$ with $S = E = \emptyset$; (b) The observation table $T_2$ 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 $\text{cex} = abab$ 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 $\text{row}(a) = \text{row}(ab)$, but $\text{row}(a.b) \neq \text{row}(ab.b)$, 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.

\[
\text{CoAc}(Q_d, \Sigma_d, \Delta_d, \delta_d, h, Q_m, q_0) \] as follows:
\[
\begin{align*}
Q_d &= \{ \text{row}(s) | s \in S \} \\
\Sigma_d &= \Sigma_o \\
\Delta_d &= \Delta \cup \{0\} \\
\delta_d(\text{row}(s), \sigma) &= \text{row}(s.\sigma) \\
h(\text{row}(s)) &= T(s, \varepsilon) \\
Q_m &= \{ \text{row}(s) | s \in S \text{ and } h(\text{row}(s)) \neq 0 \} \\
q_0 &= \text{row}(\varepsilon)
\end{align*}
\]

where $\text{CoAc}$ 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(e). $G_d(T_3)$ 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 $S$: $\text{row}(\varepsilon)$, $\text{row}(a)$, and $\text{row}(aa)$, respectively. Due to the construction procedure, the operator $\text{CoAc}$ 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 \( \text{row}(\varepsilon) \), and is shown by an entering arrow. The transition rule, \( \delta_d \), is shown by labeled, directed arrows connecting the states of the constructed automaton.

**Algorithm 1** Learning diagnoser algorithm

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

After constructing the diagnoser automaton, \( G_d(T_i) \), the diagnoser keeps running until a counterexample, \( cex \in \mathcal{L}(G_d(T_i)) \setminus P(\mathcal{L}(G)) \cup P(\mathcal{L}(G)) \cup \mathcal{L}(G_d(T_i))) \), 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 \( s = abab \in P(\mathcal{L}(G)) \setminus \mathcal{L}(G_d(T_3)) \). 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 \( S \cup S, \Sigma \). The resulting observation table, is not closed as none of the rows in \( S \) is equal to \( \text{row}(a) \) and \( \text{row}(b) \), \( a, b \in (S, \Sigma - S) \). 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 \( \text{row}(aa) = \{ F_2 \} \) 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 to the equivalence query by returning the counterexample \( cex = abab \in P(\mathcal{L}(G)) \cup \mathcal{L}(G) \). 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 \( \text{row}(ab) = \{ F_1 \} \). Therefore, the string \( ab \) is added to \( S \) in \( T_5 \) (Figure 3(f)). The new observation table, \( T_5 \), is not consistent as \( \text{row}(a) = \text{row}(ab) \), but \( \text{row}(ab, b) = \{ A_1, A_2 \} \). Therefore, \( b \) 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 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 $a_1f_1a$, then the diagnoser $G_d(T_2)$ 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 diagnoser. 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