Fault Diagnosis in Unknown Discrete Event Systems via Critical Tree

Cheng Jiang[1], Weilin Deng[1], Daowen Qiu[1]

1. Institute of Computer Science Theory, School of Data and Computer Science, Sun Yat-sen University, Guangzhou, 510006
E-mail: issqdw@mail.sysu.edu.cn (Corresponding author's address).

Abstract: In this paper, we propose an approach based on the critical tree to solve the fault diagnosis problem under the circumstances that the system model is unknown but its previous running-logs are available. First of all, we introduce the critical tree and explain its structure in detail. Secondly, we present the construction algorithm of the critical tree, and prove that it has a polynomial-time complexity both over time and space. Moreover, we show how to use the critical tree to diagnose the faults of the system. Finally, a comparative experiment is performed, and the results show that our approach has advantages both in efficiency and accuracy over Christopher's approach.

Key Words: Diagnosis, Critical Tree, Discrete Event System, Critical Observation

1 INTRODUCTION

This paper investigates the fault diagnosis problem in discrete event systems (DESs). Fault diagnosis focuses on the task of detecting and isolating the occurred fault within a finite delay [1]. This task is performed by a *diagnoser*. According to the acquisition of knowledge about the system, the diagnosis approaches can be categorized as the *white-box approaches* and the *black-box approaches* [1].

The white-box approaches (e.g., [1, 2, 3]) require full knowledge about the system to be diagnosed. That is, the diagnosers are constructed based on the exact system models. These approaches usually are precise, but costly and even infeasible sometimes, as it is quite difficult to obtain the exact models of some complicated systems.

The black-box approaches only require a group of event sequences, which are produced by the system in the past [4]. These sequences are usually called as *running-logs* in literature. The diagnosers in these approaches are constructed according to the sequences rather than the system models.Intuitively, the more sequences relies on, the more precise diagnoser could be constructed. However, the whole set of sequences, in many cases, is too large and redundancy. In other words, some sequences do not bring any useful information but just time-consume in diagnosers construction. Therefore, the problem on how to reduce the useless observations becomes more and more important with the growing of the system scale.

Recently, many approaches have been proposed to solve this problem. The optimal diagnosability approach [5] is concerned with minimizing the observable event set to ensure diagnosability in a given system. The sequential diagnosis approach [6] and dynamic observers approach [7] focus on the sensor minimization. The critical observations approach [8, 9, 10] isolates the set of fault diagnoses from the non-diagnosis ones, so that the sequences that do not provide diagnosis information are dismissed.

The notion of *critical observations* was first proposed by Christopher et al. [8], which is a set of relevant observations (observable pattern). By means of the critical observations, the set of sequences could be minimized such that the reserved sequences are all relevant to the diagnosis.

Based on their previous work in [8], Christopher et al. [9] extended the results to the event-based observation, which contains implicit information. In order to capture the implicit information, the notion of *sub-observation* was introduced in [9]. The critical observation in [9] was defined as a minimal sub-observation. Moreover, an exponential complexity algorithm for finding critical observations was proposed in [9]. In [10], a data-driven approach based on running-logs was proposed to infer the *faultly mode signatures* that was first introduced in [11]. Although the data-driven algorithm in [10] has an advantage in performance over that in [9], unfortunately it was still exponential complexity.

In this paper, it is also assumed that the DES model is unknown, and the only available information is a set of *running-logs* of G in the past. We propose a polynomial-time algorithm based on mining the critical observations by the given running-logs. For a given observation, our approach can infer its faulty mode with less time consumption, low space consumption but high accuracy. Different from Christopher's approach, in our approach, the critical observations are stored in a special data structure, called as *critical tree*.

The rest of the paper is organized as follows. Section II introduces the preliminary notions and notations, and Section III presents a formulation for the critical tree. In Section IV, we propose the construction algorithm and the query algorithm of the critical tree. Morover, we also discuss the time and space complexity of the algorithms. In

Corresponding author: Daowen Qiu (issqdw@mail.sysu.edu.cn). The work is supported in part by the National Natural Science Foundation of China under Grant 61876195 and Grant 61572532, the Natural Science Foundation of Guangdong Province of China under Grant 2017B030311011 and Grant 2017A030310583, and the Fundamental Research Funds for the Central Universities of China under Grant 17lgjc24.

978-1-7281-0106-4/19/$31.00

Section V, a comparative experiment between our approach and Christopher's approach [9] is performed. Finally, Section VI draws conclusions.

2 PRELIMINARIES

In this section, we would like to review the fault diagnosis problem of DESs, espeically the black-box diagnosis approaches. For more details, readers could refer to [1], [9], [11].

2.1 Discrete Event Systems

A discrete event system (DES) is usually modelled as a finite automaton $G = \{X, x_0, \Sigma, \delta\}$, in which X is the set of states, x_o is initial state, Σ is the set of events and $\delta : X \times \Sigma \to X$ is the partial transition function. The function δ can be extended to $X \times \Sigma^*$ by the usual manner, where Σ^* is the Kleene closure of Σ. The event set Σ can be partitioned into observable part Σ_o and unobservable part Σ_{uo}, namely, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. $\Sigma_f \subseteq \Sigma_{uo}$ is the set of faulty events. The set of runs of DES G is a prefix-closed language $L_G = \{\omega | \delta(x_0, \omega) \in X\} \subseteq \Sigma^*$. A *run* $\omega = e_1 e_2 ... e_n \in L_G$ is a finite event sequence, and the symbol ε denotes the empty sequence. The projection function $P : \Sigma^* \to \Sigma_o^*$ is defined as follows.

$$P_{\Sigma_o}(\omega) = \begin{cases} \varepsilon & \text{if } \omega = \varepsilon, \\ P_{\Sigma_o}(u)\sigma & \text{if } \omega = u\sigma \wedge \sigma \in \Sigma_o, \\ P_{\Sigma_o}(u) & \text{if } \omega = u\sigma \wedge \sigma \in \Sigma_{uo}. \end{cases} \quad (1)$$

For a run ω, only $P_{\Sigma_o}(\omega)$ can be observed. In this paper, $o = P_{\Sigma_o}(\omega)$ is called as an *observation* generated by the run ω.
In the following, for any faulty event f and a run ω, $f \in \omega$ means that the faulty event f is contained in the run ω; and $F = \Sigma_f \cap \omega$ means that $F = \{f | f \in \Sigma_f \wedge f \in \omega\}$.
In this paper, we suppose $\|\Sigma_f\| = m$. Thus, the system includes 2^m running-mode totally: a non-faulty mode and $2^m - 1$ faulty modes. Each faulty mode F_i, $\varnothing \neq F_i \subseteq \Sigma_f$, $i \in \{1, \ldots, 2^m - 1\}$, is tagged with "$T_i$", and the non-faulty mode F_0 is tagged with "T_0".

Definition 1 [10] A *running-log* l is a tuple (o, t), where o is the observation, and t is the corresponding tag.

The given set of running-logs L is supposed to be correct. Formally, for any $l = (o, t) \in L$, the following two conditions always hold: $\forall i \in \{1, \ldots, 2^m - 1\}$

1. $t = T_i \Rightarrow \exists \omega \in L_G, (P_{\Sigma_o}(\omega) = o) \wedge (\Sigma_f \cap \omega = F_i)$,
2. $t = T_0 \Rightarrow \exists \omega \in L_G, (P_{\Sigma_o}(\omega) = o) \wedge (\Sigma_f \cap \omega = \varnothing)$.

That is, the tag of a running-log is exactly the running-mode of the system

2.2 Sub-observations

Intuitively, a sub-observation is an abstraction over a observation that represents an intentional relaxation of the concrete knowledge contained in the observation o.
Before we formally give the definition of sub-observation, it is necessary to define the "hard events" and "soft events". A *hard event* is a singleton observable event, representing that the event has definitely occurred. A *soft event* is a subset of observable events, representing that the observable events and all the unobservable events (Σ_{uo}) could have occurred within any finite numbers times.

Definition 2 [9] A *sub-observation* θ is a strict time-ordered alternating sequence of soft events and hard events, such as $\theta = y_0 x_1 y_1 ... x_n y_n$, where y_i is the soft event, and x_i is the hard event, $\|\theta\| = n$ is the length of θ. The sub-observation θ can be expressed as language $L_\theta = (y_0 \cup \Sigma_{uo})^* x_1 (y_1 \cup \Sigma_{uo})^* ... x_n (y_n \cup \Sigma_{uo})^*$.

For any observation o and a sub-observation θ, we say o *matches* θ, denoted as $o \in \theta$, if $o \in P_{\Sigma_o}(L_\theta)$. We use $\mathbb{O}(o)$ to denote the set of sub-observations that can be matched by o, and $\mathbb{O}$ to denote all the sub-observations that can be matched by any observation of the system. For instance, given an observation $o = bbdacaa$, it matches the sub-observation $\theta = \{b, d\}a\varnothing c\{a\}$, in which $y_0 = \{b, d\}$, $x_1 = a$, $y_1 = \varnothing$, $x_2 = c$, $y_2 = \{a\}$. Thus, we have $\|\theta\| = 2$, $o \in \theta$, and $\theta \in \mathbb{O}(o)$.

Definition 3 [9] For two sub-observations $\theta, \theta' \in \mathbb{O}$, θ' is said to be more *abstract* than θ, denoted as $\theta' \preceq \theta$, if and only if $\forall o \in \theta$, we have $o \in \theta'$. If $\exists o \in \theta', o \notin \theta$, then we said θ' is *strict abstract* than θ, denoted as $\theta' \prec \theta$.

For example, $\theta' = (\{b, d\}, a, \{a, c\})$ is more abstract than $\theta = (\{b, d\}, a, \varnothing, c, \{a\})$, that is $\theta' \preceq \theta$.
Obviously, for any observation, the most abstract sub-observation is always Σ^*, and for the observation $o = e_1 ... e_n$, the least abstract sub-observation is $sub(o) = \varnothing e_1 \varnothing ... e_n \varnothing$. Thus, if $\theta \preceq sub(o)$, then $o \in \theta$.

Definition 4 [10] A *child* of sub-observation θ is a strict sub-observation θ' of θ such that no sub-observation sits between θ' and θ, that is,

$$(\theta' \prec \theta) \wedge (\nexists \theta'' \in \mathbb{O} : \theta' \prec \theta'' \prec \theta).$$

We use $children(\theta)$ to denote the child set of θ. Moreover, θ is said to be the *parent* of θ'.

2.3 Fault Diagnosis

A diagnosis of a sub-observation θ, denoted as $\Delta_L(\theta)$, is the collection of all possible faulty sets:

Definition 5 Given a set of running-logs $L = \{l_1, l_2, ..., l_n\}$, $l_j = (o_j, t_j)$, $j \in \{1, \ldots, n\}$, $\Delta_L(\theta)$ is defined as follows.

$$\Delta_L(\theta) = \bigcup_{j=1}^{n} \{F_i | t_j = T_i \wedge o_j \in \theta\}, i \in \{1, \ldots, 2^m - 1\}. \quad (2)$$

In particular, if there does not exist such faulty sets, then the diagnoser returns $\varnothing$.
A sub-observation θ is said to be *sufficient* for observation o, if there exist a running-log $l = (o, t)$ and a faulty mode F_i, such that $\Delta_L(\theta) = F_i$ and $T_i = t$.
If the sub-observation is the sufficient but minimum abstract, then we said it is *critical*.

Definition 6 Given a log $l = (o, t) \in L$, a sub-observation θ, $o \in \theta$, is said to be *critical* for o, if and only if θ is sufficient for o, and there exists a child θ' that is not sufficient, that is:

$$\Big((\exists F_i \subseteq \Sigma_f)\Delta_L(\theta) = F_i \wedge T_i = t\Big) \wedge \Big(\exists \theta' \in children(\theta) : (\Delta_L(\theta') \neq \Delta_L(\theta))\Big). \quad (3)$$

3 CRITICAL TREE

The goal of our algorithm is to conjecture the faulty mode of an observation according to a set of running-logs. In order to accomplish the task, we need to construct a special structure named *critical tree*. The critical tree is a multi-way tree, whose inner node is labeled by a sub-observation and leaf node is labeled by a running-log. Besides, for a inner node, the sub-observation labeled in its parent node is more abstract than the sub-observation labeled in itself. Moreover, if its child node is a leaf node, then the sub-observation is a critical observation of the child node's log observation. We give the formal definition of *critical tree* as follows.

Definition 7 Assume the set of running-logs is L, the *critical tree* $CtT(L)$ is a multi-way tree

$$CtT(L) = (Q, S, \psi, \phi, q_0),$$

in which

- $Q = Q_I \cup Q_L$ is a finite set of nodes. Q_I is the set of inner nodes, and each inner node is labeled with a sub-observation θ. Q_L is the leaf node set, and each leaf node is labeled with a running-log $l \in L$;
- $S \subseteq \Sigma_o^*$ is a set of observations of the system;
- $\psi : Q_I \to \mathbb{O}$ is a label function that extracts the sub-observation from the inner node;
- $\phi : Q_L \to L$ is a label function that extracts the running-log from the leaf node;
- $q_0 \in Q_I$ is the root node that is labeled with the sub-observation Σ^*.

For each two nodes $q, q' \in Q$ in the tree, the following four conditions need to be satisfied. 1.*hierarchy*: if $q, q' \in Q_I$ and q is the parent node of q', then $\psi(q)$ is a child of $\psi(q)$; 2.*sufficiency*: if $q \in Q_I, q' \in Q_L$ and q is the parent node of q', then $\psi(q)$ is critical for $\phi(q')$; we call q is the *critical node* of q'; 3.*incompatibility*: if $q, q' \in Q_I$ and q is the sibling-node of q', then $\theta \not\leq \theta'$ and $\theta' \not\leq \theta$; 4.*similarity*: if $q, q' \in Q_L$ and q is the sibling-node of q', then $\phi(q)$ and $\phi(q')$ have the same faulty mode and critical observation.
For any inner node q_I in the tree, its ancestor is labeled with the more abstract sub-observation, so that it has less exact diagnosis information. Its siblings are in the same abstraction, but each sibling preserves different diagnosis information on the basis of its parent's.
For any path $P = q_0 \to q_1 ... \to q_n \to q_l$ from the root node q_0 to the leaf node q_l in the tree, the observation $o \in \psi(q_l)$ matches all sub-observations labeled in the inner node $\psi(q_j)$. That is, $\forall j \in \{0, 1, ..., n\}, o \in \phi(q_j)$. Moreover, $\psi(q_n)$ is critical observation for o.
Note that any two running-logs that have the same critical observation will have the same faulty mode. That is, one critical node can acts as many leaf node's parent. However, two or more leaf nodes that have different critical node may be labeled with the same running-log.

3.1 Example

Suppose $\Sigma = \{a, b, c\}$ and $L = \{(aab, F1), (aaab, F1), (aac, F2)\}$, then the critical tree is shown in Figure 3.1.

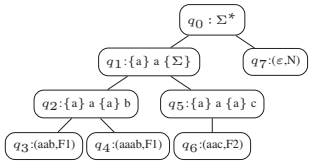


Figure 1: A critical tree for the set of running-logs $L = \{(aab, F1), (aaab, F1), (aac, F2)\}$

3.2 Fault Diagnosis on Critical Tree

We can acquire a set of critical observations in a critical tree for all critical node is critical. The diagnoser base on critical tree is proposed as follows.

Definition 8 Given a set of running-logs L and its critical tree $CtT(L) = (Q, S, \psi, \phi, q_0)$, for any future observation o produced by the system, if there exists a path $P = \{q_0, q_1, ..., q_n, q_l\}$:$\forall j \in \{0, 1, ..., n\}, o \in \psi(q_j)$ and $\phi(q_l) = (o, t)$, then Δ_C returns the diagnosis:

$$\Delta_C(o) = \bigcup\{F_i | (\exists P = \{q_0, q_1, ..., q_n, q_l\}) \wedge (\forall j \in \{0, 1, ..., n\}, o \in \psi(q_j)) \wedge (t \in \phi(q_l) \wedge t = T_i)\}. \quad (4)$$

For convenience, we said the observation o matches path P, simply as $o \in P$. Finally, we said the diagnoser returns right result if and only if the critical tree is correct.

Definition 9 A critical tree $CtT(L)$ is *correct* if the diagnosis of all leaf log observations is correct, that is $\forall q_l \in Q_L, \phi(q_l) = (o, T_i) \Rightarrow \Delta_C(o) = F_i$.

Note that we do not require the diagnose path must be the path that is from the root to the diganosed leaf node, in contrast, we just need the diagnose result is right. The goal of our algorithm is to ensure the constructed critical tree correct.

4 CTT ALGORIHTM

In this section, we present a new algorithm to infer the faulty mode of an observations using the critical tree. First of all, we propose the construction algorithm of the critical tree. Secondly, we show how to acquire the faulty mode by querying the critical tree. Finally, we present an illustrative example.

Procedure 1 $update(CtT, l)$: Updating the critical tree

Input: $CtT = (Q, S, \psi, \phi, q_0)$,$l = (o, t)$
Output: CtT

function UPDATE(CtT, l)
  set $n = q_0$
  while $n \in Q_I$ **do**
    set $\theta = \psi(n)$
    if $o \in \theta$ **then**
      set $n = child(n)$
    else
      set $n = sibling(n)$
    end if
  end while
  if $n \in Q_L$ **then**
    set $n_p = parent(n)$
    while $n \in Q_L$ **do**
      set $(o_n, t_n) = \phi(n)$
      if $o = o_n$ **then**
        if $t \neq t_n$ **then**
          $removeConflicting(n_p, l)$
        end if
        return CtT
      end if
      set $n = sibling(n)$
    end while
    if $t = t_n$ **then**
      set $n_L = createLeafNode(l)$
      $link(n_p, n_L)$
    else if $t \neq t_n$ **then**
      set $\mathbb{O} = genCritical(\theta, o, o_n)$
      $insertIntoTree(n_p, \mathbb{O}, l)$
      set $\mathbb{O}_n = genCritical(\theta, o_n, o)$
      $insertIntoTree(n_p, \mathbb{O}_n, l_n)$
    end if
  else
    set $n = previousSibling(n), n_p = parent(n)$
    set $\theta = \psi(n_p)$
    set $\mathbb{O} = genCritical(\theta, o, \phi(child(n)))$
    $insertIntoTree(n_p, \mathbb{O}, l)$
  end if
  return CtT
end function

Algorithm 1 Construct critical tree

Input: a log set L
Output: critical tree CtT

$initialize(CtT)$
$L_C = removeConflict(L)$
for $l \in L_C$ **do**
  $update(CtT, l)$
end for
return CtT

4.1 Critical Tree Construction

The critical tree is initialized with two nodes, one is the root labeled by Σ^*, the other is the leaf node labeled by (ε, N). Given a set of running-logs, the whole critical tree could be constructed by continuously updating itself with handling each running-log. The whole construction process for critical tree is shown in Algorithm 1.

Note that before doing the updating, removing the conflicting running-logs is necessary. The conflicting logs is defined as follows.

Definition 10 Two log $l = (o, t)$ and $l' = (o', t')$ are said to be *conflicting running-logs* if $o = o' \wedge t \neq t'$.

4.1.1 Update critical tree

The updating procedure is show in Procedure 1. The procedure includes two steps: accepting a running-log firstly, and then placing the running-log to the right location of the critical tree. Firstly, we need to find a path that matches for o until the query node n is either a leaf node or it does not exist. Suppose that n is labeled by $l_n = (o_n, t_n)$, and $\theta = \psi(n_p)$ is its critical observation.

If $n \in Q_L$, we need to consider the following three conditions:

1. If $\exists n \in children(n_p), o = o_n$, it means the observation o already exists in CtT. Furthermore, if $t = t_n$, we ignore it; otherwise, the log is a conflicting log, hence we perform $removeConflicting$ to remove all conflicting log. That is, if θ is critical only for o, then we remove it either; otherwise we perform $genCritical$ to generate new critical observations, as θ is not sufficient now.

2. If $t = t_n$, then θ remains sufficient for o, and we add a new leaf node labeled with l, as a child node of n_p.

3. If $t \neq t_n \wedge o \in \theta$, then θ is not sufficient now, hence, we need to perform the $genCritical$ procedure to generate new critical observations from θ. Executing the $genCritical$ procedure could ensure that all new sub-observations are the parent of θ, and each two of them are incompatible. After doing the generating, The $insertIntoTree$ procedure inserts all new inner nodes into the tree, as the children of n_p, and all these child leaf nodes are labelled with the running-log.

On the other hand, if $n \notin Q$, it means CtT lacks of the critical observation of l, and therefore, we perform the procedures $genCritical$ and $insertIntoTree$, respectively. Note that before performing the two function, it is necessary to make trace n back to an existent node.

Proposition 1 The procedure $update$ keeps hierarchy, sufficiency, incompatibility and similarity of the critical tree.

4.1.2 Generate new critical observation

The procedure $generateCritical(\theta, o, o_n)$ is shown in procedure 2. The main idea is finding a new hard event sequence h_{new}, which is a subsequence that occurred in the observation o but disappeared in the old observation o'. h_{new} can be extended to a sub-observation by filling the soft event with the events that occurred between each two adjacent hard events in o.

The solution of finding h is the same as that of finding a shortest uncommon subsequence between o and o', and the later algorithm is polynomial-time complexity. The function $findNewSeq$ searches all shortest uncommon

Algorithm 2 Query critical tree

Input: $CtT(L) = (Q, S, \psi, \phi, q_0)$,$o$
Output: faulty mode t

```
set q = q_0
while q ∈ Q_I do
    if o ∈ ψ(q) then
        set q = child(q)
    else
        set q_s = sibling(q)
        while q_s ∉ Q do
            set q = parent(q)
            if q ∉ Q then
                break
            end if
            set q_s = sibling(q)
        end while
        set q = q_s
    end if
end while
if q ∈ Q_L then
    set t ∈ φ(q)
else
    set t = T_0
end if
return t
```

Procedure 2 $genCritical(\theta, o, o_n)$: Generating critical observations

```
1: function GENCRITICAL(θ, o, o')
2:     set h = (x_1, x_2, ..., x_m), Θ = ∅
3:     set (f, l) = findPos(o, h), (f', l') = findPos(o', h)
4:     for all j ∈ {0, 1, ..., m} do
5:         set o_j = e_{f_j} e_{f_j+1} ... e_{l_j}, o'_j = e_{f'_j} e_{f'_j+1} ... e_{l'_j}
6:         set E_new = findNewSeq(o_j, o'_j)
7:         for all e ∈ E_new do
8:             set h_new = insertNewHard(h, e, j)
9:             set Θ = Θ ∪ fillSoft(o, h_new)
10:        end for
11:    end for
12:    return Θ
13: end function
```

subsequences E_{new}. After searching, $insertNewHard$ inserts e_{new} into h, then a new hard event sequence h_{new} is acquired. Finally, function $fillSoft$ extends h_{new} to a new sub-observation θ_{new}. By doing so, θ_{new} could satisfy the condition $o \in \theta_{new} \wedge o' \notin \theta_{new}$.

Proposition 2 θ is a child of θ_{new}, and if θ is a critical observation before $udpate$, then θ_{new} is a critical observation.

Proposition 3 Each two sub-observation $\theta_1, \theta_2 \in \Theta$ after $genCritical$ satisfies $\theta_1 \not\leq \theta_2$ and $\theta_2 \not\leq \theta_1$.

Proposition 4 Assume $\|L\| = n$, $\|o_{max}\| = k$ and $\|\Sigma\| = l$. The time complexity and space complexity of Algorithm 1 are $O(nk + lk^2)$ and $O(nk)$, respectively.

Theorem 1 The critical tree constructed by Algorithm 1 is correct. That is, $\forall q_l \in Q_L, \phi(q_l) = (o, T_i) \Rightarrow \Delta_C(o) = F_i$.

4.2 Inferring the faulty mode using the critical tree

The critical tree querying algorithm is shown in Algorithm 2. According to the algorithm, we could obtain the faulty mode for a given observation o.

The query algorithm uses the deep-first searching mechanism. When querying an inner node q_I, we check if the observation matches the corresponding sub-observation ($o \in \phi(q_I)$). If it matches, queries its first child node next, else query its next sibling node next. When querying a leaf node, according to the definition 8, we would find a path $P \wedge o \in P$. Consequently, the algorithm returns the faulty mode t labeled in the leaf node. If it cannot find any critical observation that matches o, the the algorithm returns T_0. It means the system is running in normal mode.

4.3 Example

Given a set of running-logs $L = \{(cccd, T_1), (aba, T_1), (aba, T_0), (ccae, T_2)\}$, we consider how to construct the critical tree according to Algorithm 1 in the following. Firstly, we remove the conflicting logs (aba, T_1) and (aba, T_0). Consider the running-log $(cccd, T_1)$, the trace $cccd \in \Sigma^*$, hence we query its child node next. Since its child is a leaf node labeled by (ε, T_0) and $T_1 \neq T_0$, new critical observations shoud be generated from $cccd$. By comparing $cccd$ and ε, sub-sequences c and d are found. Then by inserting c and d into Σ^* and extending the new hard event sequences from $cccd$, two new critical observation $\{c\}c\{cd\}$ and $\{c\}d$ are obtained. Finally, we insert new critical nodes and their leaf nodes into the critical tree, as shown Figure 2-(a). Repeat the same steps for the second running-log $(ccae, T_2)$, we obtain the final critical tree, as shown in Figure 2-(b).

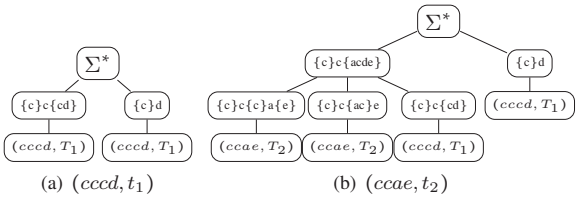


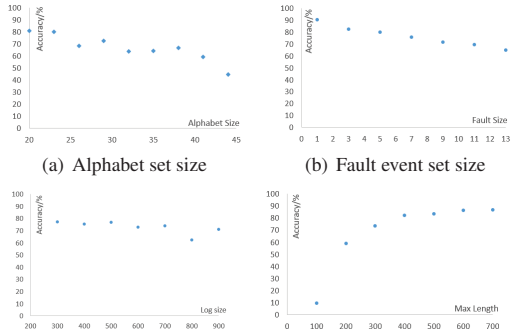
Figure 2: An example for $L = \{(cccd, Tt_1), (aba, T_1), (aba, T_0), (ccae, T_2)\}$

In the following, we show how to obtain the diagnosis of the observation $cccd$ via querying the critical tree shown in Figure 2-(b). Firstly, $cccd \in \Sigma^*$, so we check its child node $\{c\}c\{acde\}$ and also obtain positive result. Then, we continue to check the child nodes of $\{c\}c\{acde\}$, and negative results from $\{c\}c\{c\}a\{e\}$ and $\{c\}c\{ac\}e$, but obtain positive result for the its last child node $\{c\}c\{cd\}$. Since the child of $\{c\}c\{cd\}$ is a leaf node with a running-log tagged as T_1, we obtain the diagnosis result: T_1 finally.

5 EXPERIMENT

In this section, we introduce two simple experiments, one for demonstrating the accuracy of our approach, the other for comparing our approach with Christopher's approach [9]. The running-logs adopted in our experiments are generated by a random deterministic finite automaton, and

its state number is a constant, as the complexity of our algorithm only relies on $\|L\|$, $\|o_{max}\|$, $\|\Sigma\|$ and $\|\Sigma_f\|$.
Figure 3 shows the accuracies of our approach. These experiments are performed with the condition of $\|L\|$ = 600, $\|o_{max}$ = 300$\|$, $\|\Sigma\|$ = 30 and $\|\Sigma_f\|$ = 10, however, one parameter may be changed for analysis.



(a) Alphabet set size (b) Fault event set size

(c) Log set size (d) Maximal length

Figure 3: Experiment results of the accuracy on CtT

Figure 3-(a) reveals the fact that for a smaller $\|\Sigma\|$, the knowledge about the system of observation is more rich and precise than a larger $\|\Sigma\|$. As Figure 3-(b) shown, with $\|\Sigma_f\|$ increasing, the accuracy decreases fast. The fact meets our intuitiveness, as the number of the faulty modes *exponentially* increases with the increasing of $\|\Sigma_f\|$. The Figure 3-(c) shows an interesting result that the accuracy decreases slightly when $\|L\|$ increases, it may be due to the over-saturation of running-logs. Figure 3-(d) illustrate the fact that a longer observation implies more information than a shorter one.
Figure 4 shows the comparison results between our approach (CtT) with Christophers approach (CO) [9]. The space-consume, time-consume and accuracy between CO and CtT are compared. The abscissa values are the form of $\|\Sigma\|\backslash\|\Sigma_f\|\backslash\|o_{max}\|\backslash\|L\|$. Here, we denote the space size as the number of sub-observations that are produced along the algorithm running.

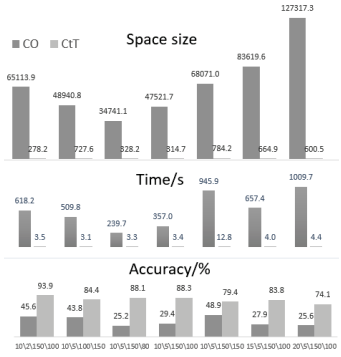


Figure 4: Experiment results of comparing CO and CtT. The abscissa values are the form of $\|\Sigma\|\backslash\|\Sigma_f\|\backslash\|o_{max}\|\backslash\|L\|$

Except for experiment, the time complexity of CO approach is $\Theta(\|L\|\|o_{max}\|^2\|\Sigma_f\|^2)$, and the worst case is exponential time complexity while CtT approach is in polynomial time complexity. Different from the sub-observation generating method in CO (by inserting each event into each soft event), our method uses the new hard event sequences to perform this task, which can avoid generating a lot of useless sub-observations. This is the main reason such that our approach (CtT) has advantages both in efficiency and accuracy over Christophers approach (CO).

6 CONCLUSION

In this work, we have presented a polynomial-time complexity approach for fault diagnosis problem based on the critical tree under the circumstances that the system model is unknown but its previous running-logs are available. In our approach, the critical tree has been constructed to store all critical observations. With the critical tree, we could obtain the critical observation by only comparing the runs with the same faulty mode instead of all runs. Thus, given observation, the critical tree can infer its faulty mode as the tree have similar running-logs producing the same fault.

Acknowledgements

The authors would like to thank Dr. Rong Su for significant discussion that helped us improve this paper.

REFERENCES

[1] M. Sampath, R. Sengupta, S. Lafortune, et al., Diagnosability of discrete-event systems, IEEE Transactions on Automatic Control, vol.40, No.9, 1555-1575, 1995.

[2] E. Fabre, L. Hlout, E. Lefaucheux, et al., Diagnosability of repairable faults, Discrete Event Dynamic Systems, Vol.28 No.2, 183-213, 2018.

[3] O. Contant, S. Lafortune, D. Teneketzis, Diagnosis of intermittent faults, Discrete Event Dynamic Systems, Vol.14, No.2, 171-202, 2004.

[4] F. Fessant, F. Clrot, An efficient SOM-based pre-processing to improve the discovery of frequent patterns in alarm logs, Conference on Data Mining (DMIN), Vol. 6, 277, 2006.

[5] L. B. Briones, A. Lazovik, P. Dague, Optimal observability for diagnosability, Nineteenth International Workshop on Principles of Diagnosis, 31-38, 2008.

[6] J. de Kleer, Using crude probability estimates to guide diagnosis, Artificial Intelligence, Vol.45, No.3, 381-391, 1990.

[7] F. Cassez, S. Tripakis, Fault diagnosis with dynamic observers, 2008 IEEE 9th International Workshop on Discrete Event Systems (WODES), 212-217, 2008.

[8] C. J. Christopher, M. O. Cordier, A. Grastien, Critical observations in a diagnostic problem, 2014 IEEE 53rd Annual Conference on Decision and Control (CDC), 382-387, 2014.

[9] C. J. Christopher, A. Grastien, Formulating event-based critical observations in diagnostic problems, 2015 IEEE 54th Annual Conference on Decision and Control (CDC), 4462-4467, 2015.

[10] C. J. Christopher, Y. Pencol, A. Grastien, Inference of fault signatures of discrete-event systems from event logs, Proceedings of the 19th International Workshop on Principles of Diagnosis, 219-233, 2017.

[11] M. O. Cordier, L. Trav-Massuyes, X. Pucel, Comparing diagnosability in continuous and discrete-event systems, Proceedings of the 17th International Workshop on Principles of Diagnosis, 55-60, 2006.