Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 263 (2025) 854–863

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry Sciences and Computer Science Innovation

Timely Diagnosis of Discrete-Event Systems

Gianfranco Lamperti*, Marina Zanella

Department of Information Engineering, University of Brescia, 25123 Brescia, Italy

Abstract

A major problem afflicting decision support systems based on automated diagnosis is a possibly large number of diagnoses explaining the observations, which may jeopardize the task of diagnosticians, owing to the cognitive overload raised by an overwhelming number of faulty scenarios to examine before undertaking effective recovery actions. This criticality is exacerbated when the recovery actions are expected to be performed automatically by an artificial agent in real-time, even in the order of milliseconds, like in an autonomous vehicle or in a defense system. To make diagnosis in the specific domain of discrete-event systems (DESs) viable in critical, real-time applications, a TIMELY DIAGNOSIS ENGINE is presented, which is based on a parsimony principle: given a set of diagnoses implied by the observations, only minimal diagnoses are elicited as candidates, on the grounds that each minimal diagnosis is more likely than any non-minimal diagnosis that is a superset of it. Experimental results indicate that the search space of the diagnosis engine is reduced considerably, owing to the pruning of the trajectories of the DES that are not bound to generate minimal diagnoses, thereby resulting in a considerable reduction not only in the number of candidates but also in processing time.

© 2025 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer review under the responsibility of the scientific committee of the International Conference on Industry Sciences and Computer Science Innovation

Keywords: decision support systems; model-based reasoning; minimal diagnosis; real-time diagnosis; discrete-event systems; uncertainty

1. Introduction

Automated diagnosis is still a challenging task to AI. In 1987, two seminal works were published in a same issue of the *Artificial Intelligence* journal, one formalizing a theory of diagnosis [45], the other [28] presenting a *general diagnostic engine* adhering to that theory. The novel approach, later called *consistency-based* diagnosis, was initially conceived for static systems, such as combinational circuits [26, 28, 20, 13]. According to it, in order to generate a *diagnosis (output)*, namely a set of faulty components, no collection of rules, derived from human expertise and linking symptoms (observations) to their causes (faults), is needed. What is necessary is instead a model of the normal (correct) behavior of (each component of) the system under analysis, which is expressed in a logical formalism. Given a collection of observations of the system, typically some input/output measurements, a number of *conflict sets* are

* Gianfranco Lamperti. Tel.: +39-030-371-5491 ; fax: +39-030-380-014.
E-mail address: gianfranco.lamperti@unibs.it

1877-0509 © 2025 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer review under the responsibility of the scientific committee of the International Conference on Industry Sciences and Computer Science Innovation
10.1016/j.procs.2025.07.103

drawn, each set involving some components, such that assuming that all of the components in the set are normal does not preserve the logical consistency between the collection of observations and the system model. Different diagnoses, called *candidates*, are then enumerated based on these conflict sets. Since conflict sets may lead to a (very) large number of different diagnoses, only *subset-minimal* diagnoses are considered as candidates: a diagnosis is subset-minimal (or simply *minimal* hereafter) when it is not a superset of any other diagnosis. In consistency-based diagnosis, any superset of a minimal diagnosis is a diagnosis itself, hence minimal diagnoses are representatives of all diagnoses. Minimal diagnoses have been shown to be the (minimal) hitting sets of the (minimal) conflict sets. Since the generation of both the conflict sets and the hitting sets are NP-hard problems, several techniques were proposed to make these two steps more efficient, possibly at the expense of completeness [1, 27, 48].

Computing diagnoses by processing a model that is specific to the considered system by means of a domain-independent reasoning engine, as suggested in [45], became the foundation of a new paradigm, called *model-based diagnosis* [21], which encompasses not only consistency-based diagnosis but also *abduction-based* diagnosis [39]. While the models adopted in [45] are *weak*, as they describe the normal behavior only, abduction-based diagnosis requires *strong* models, i.e. models that describe not only the normal behavior but also (some) faulty behavioral modes. Abduction-based diagnosis may be a better choice when dynamical systems are considered, especially when discrete-event systems (DESs) come into play [7], theses being systems endowed with a finite number of states that evolve over qualitative time based on a finite number of (discrete) events.

Apart from a few exceptions, including [41], abduction-based diagnosis of DESs assumes a *complete* model of the system, involving all behaviors, both normal and faulty. The model of a DES may be a Petri net, as in [23, 6, 3, 8], or, more typically, a network of finite communicating automata [5], like in [2, 11, 40, 17, 24, 29, 36] and in this paper. Each component is modeled as an automaton that can react to events either generated by other components or coming from outside, thereby possibly causing the generation of new events. In the component model, a state transition may be qualified as faulty, in which case a *fault* is associated; also, a component transition may be observable, in which case an *observation* is associated. A DES represented by (one or several) communicating automata is either synchronous or asynchronous: if synchronous, such as in [46, 47], component transitions may be triggered in parallel, based on the same event, whereas, if asynchronous, such as in [31, 33, 38, 36, 37] and in this paper, only one component transition at a time can occur. Assuming asynchronism, a *trajectory* of the DES is a sequence of component transitions from the initial state of the DES to a final state. A trajectory is perceived from the outside of the DES as a sequence of observations associated with the observable transitions of the trajectory, called a *temporal observation*. The diagnosis engine can compute the candidates by finding out the trajectories of the DES generating a temporal observation that has been perceived. Diagnosis of DESs can be either *a posteriori* or *monitoring-based*. When a posteriori, the engine is activated after the reception of a complete temporal observation. In monitoring-based diagnosis, instead, the engine is expected to generate the candidates upon the reception of each new observation, as is in the fundamental work by Sampath et al. [46, 47], where the notion of *diagnosability* of a DES is coined and a *diagnoser* is defined in order to support the online diagnosis task efficiently.

Since the number of candidates relevant to a DES may be overwhelming for the human diagnostician who has to make a decision, possibly under stringent time constraints, or even for an artificial real-time agent, and given the higher probability of a minimal diagnosis with respect to any diagnosis that is a superset of it[1], computing *minimal* diagnoses of DESs is proposed in [49], where the notion of *minimal diagnosability* is presented, along with a *minimal diagnoser*. The problem in the construction of a minimal diagnoser for a real DES, however, is the need to first generate the space of the DES, whose number of states is exponential in the number of components. This is why here, and more generally in the *active system approach* [32, 36] from which this work stems, we assume the unavailabilty of the DES space, which is only instrumental to the formalization, but never materialized. The contribution of this paper is a monitoring-based TIMELY DIAGNOSIS ENGINE for DESs, which generates minimal diagnoses without requiring the diagnosability of the DES, nor the support of any diagnoser. Experimental results indicate that the search space of

[1] Assuming that faults in a DES are reciprocally independent, following [28], the probability of a candidate is the product of the individual probabilities of each fault in the candidate to occur and each remaining fault not to occur. If the (possibly unknown) probability value is (sensibly assumed to be) less than 0.5 for every fault, each minimal diagnosis is more probable than any of its supersets, both in case the probabilities of faults are equal to each other or different from each other. This additionally implies that the (possibly not unique) most probable diagnosis is a minimal diagnosis.

the algorithm is reduced considerably, owing to the pruning of the trajectories that do not generate minimal diagnoses, thereby resulting not only in a far lower number of candidates but also in a substantially shorter processing time.

2. Background

The topology of a distributed, asynchronous DES is a network of *components* that are modeled as finite communicating automata and are connected to other components through *links*. When the DES starts being operated, each component is in its initial state and links are empty. The occurrence of an event outside the DES may trigger a state transition in a component, which may generate events towards other components via links. This results in a cascade of state transitions, called a *trajectory*, that moves the DES from its initial state to a new system state. The trajectories of a DES $\mathcal{X}$ are confined to a *space*, which is itself a finite automaton, namely $Space(\mathcal{X}) = (\Sigma, X, \tau, x_0)$, where the alphabet Σ is the set of component transitions, X is the set of (system) states, where a state is a pair of a tuple of states of components and a tuple of the events within links, τ is the transition function mapping a state and a component transition into a new state, and x_0 is the initial state.

Example 1. Outlined on the left of Figure 1 is a DES, called $\mathcal{W}$ (*watcher*), which is designed to protect a power transmission line from short circuits, typically caused by lightnings. It embodies two components, a protection p and a breaker b, and a link connecting p to b. A short circuit striking the line is perceived by the protection as a drop in voltage: if so, the protection commands the breaker to open in order to get the short circuit extinguished, in which case the protection commands the breaker to close in order to restore the electric power. The models (communicating automata) of both p (top) and b (bottom), which are displayed on the center of the figure, involve two states (represented as ellipses), with four and eight transitions, respectively (represented as arrows between states). Each arc is marked with the name of the corresponding transition. Component transitions are described in Table 1 (first and second columns). Each component transition from a state s to a state s' that is triggered by an input event e and generates a set of output events E is denoted by a triple $\langle s, (e, E), s' \rangle$. If e is the *empty* event ε, it means that the transition is triggered by an external event: from the DES point of view, the transition is *spontaneous*. For instance, transition $p_1 = \langle normal, (\varepsilon, \{op\}), shorted \rangle$ of the protection is triggered by an external event (drop in line voltage), generates the single event *op* (to open the breaker), and moves the protection from state *normal* to state *shorted*. Note how all transitions of the breaker are triggered by an event generated by the protection (either *op* or *cl*), and do not generate output events ($E = \emptyset$).

The space of $\mathcal{W}$ is shown on the right of Figure 1, where states are renamed $0 \ldots 7$, with 0 being the initial state. Each state is marked with a pair of states of components p and b, along with the possibly empty (ε) event within the link. A trajectory of $\mathcal{W}$ is, for instance, $T = [p_3, b_5, p_1, b_3, p_4, b_3, p_2, b_5]$. Owing to cycles, the space contains an unbounded number of possible trajectories of $\mathcal{W}$.

To support the diagnosis of a DES $\mathcal{X}$, we specify both the *observability* and the *abnormality* of $\mathcal{X}$ by means of a *map*, namely $Map(\mathcal{X})$, which is a set of triples (t, o, f), where t is a component transition, o is a (possibly empty) observation,

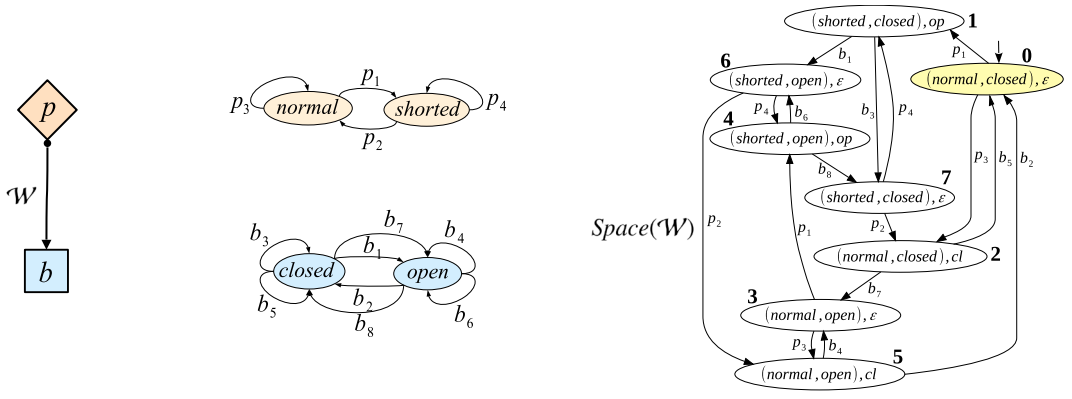

Figure 1: DES $\mathcal{W}$ (left), component models (communicating automata) of protection p and breaker b (center), and $Space(\mathcal{W})$ (right).

Table 1: Description of transition actions for protection p and breaker b in DES $\mathcal{W}$, along with observations and faults in $Map(\mathcal{W})$.

transition	*action*	*obs*	*fault*
$p_1 = \langle normal, (\varepsilon, \{op\}), shorted\rangle$	p reacts normally to a short circuit by generating the open event	*prt*	ε
$p_2 = \langle shorted, (\varepsilon, \{cl\}), normal\rangle$	p reacts normally to a short circuit extinction by generating the close event	*prt*	ε
$p_3 = \langle normal, (\varepsilon, \{cl\}), normal\rangle$	p reacts abnormally to a short circuit by generating the close event	ε	*u*
$p_4 = \langle shorted, (\varepsilon, \{op\}), shorted\rangle$	p reacts abnormally to a short circuit extinction by generating the open event	ε	*v*
$b_1 = \langle closed, (op, \emptyset), open\rangle$	b reacts normally to the open event by opening	*brk*	ε
$b_2 = \langle open, (cl, \emptyset), closed\rangle$	b reacts normally to the close event by closing	*brk*	ε
$b_3 = \langle closed, (op, \emptyset), closed\rangle$	b reacts abnormally to the open event by remaining closed	ε	*y*
$b_4 = \langle open, (cl, \emptyset), open\rangle$	b reacts abnormally to the close event by remaining open	ε	*z*
$b_5 = \langle closed, (cl, \emptyset), closed\rangle$	b reacts normally to the close event by remaining closed	*brk*	ε
$b_6 = \langle open, (op, \emptyset), open\rangle$	b reacts normally to the open event by remaining open	*brk*	ε
$b_7 = \langle closed, (cl, \emptyset), open\rangle$	b reacts abnormally to the close event by opening	*brk*	*w*
$b_8 = \langle open, (op, \emptyset), closed\rangle$	b reacts abnormally to the open event by closing	*brk*	*q*

and f is a (possibly empty) fault. Specifically, if $o \neq \varepsilon$, then t is *observable*, otherwise t is *unobservable*; likewise, if $f \neq \varepsilon$, then t is *faulty*, otherwise t is *normal*. Based on $Map(\mathcal{X})$, each trajectory T of DES $\mathcal{X}$ is associated with a *temporal observation*, which is the sequence of the observations associated with the observable component transitions in T, namely $Obs(T) = [\, o \mid t \in T, (t, o, f) \in Map(\mathcal{X}), o \neq \varepsilon \,]$. A trajectory T *conforms* with a temporal observation O iff $Obs(T) = O$. $Map(\mathcal{X})$ also associates T with a *diagnosis*, which is the set of faults associated with the component transitions in T, namely $Dgn(T) = \{ f \mid t \in T, (t, o, f) \in Map(\mathcal{X}), f \neq \varepsilon \}$. The *diagnosis set* $\mathcal{D}$ of O is the set of diagnoses of the trajectories of $\mathcal{X}$ conforming with O, namely: $\mathcal{D}(O) = \{Dgn(T) \mid T \in Space(\mathcal{X}), O = Obs(T)\}$. Let O_i denote a nonempty prefix $[o_1, \ldots, o_i]$ of O, $i \geq 1$. The *temporal diagnosis set* $\mathbf{D}$ of O is the sequence of diagnosis sets $\mathcal{D}(O_i)$, $i \geq 1$, namely: $\mathbf{D}(O) = [\mathcal{D}(O_1), \mathcal{D}(O_2), \ldots]$.

Example 2. Assuming a set of observations $\{prt, brk\}$ and a set of faults $\{u, v, y, z, w, q\}$, $Map(\mathcal{W})$ is embedded in Table 1, where for each component transition, both an observation (in blue) and a fault (in red) are associated. Only one observation is defined for both the protection and the breaker, namely *prt* and *brk*, respectively. Accordingly, transition p_1 is observable and normal, p_3 is unobservable and faulty, and b_7 is both observable and faulty. Since the same observation is associated with several transitions, uncertainty remains in the determination of the component transition based solely on the observation occurred. With reference to trajectory $T = [p_3, b_5, p_1, b_3, p_4, b_3, p_2, b_5]$, we have $Obs(T) = [brk, prt, prt, brk]$ and $Dgn(T) = \{u, v, y\}$. Based on $Space(\mathcal{W})$, we can find that the diagnosis set of $Obs(T)$ includes six diagnoses: $\{u, y\}$, $\{u, v, y\}$, $\{u, y, w\}$, $\{u, v, y, w\}$, $\{u, y, z, w\}$, and $\{u, v, y, z, w\}$. Note how the diagnosis set involves the diagnosis $\{u, v, y\}$, namely $Dgn(T)$, but, owing to uncertainty, five additional diagnoses are embodied in that set. The temporal diagnosis set of O is $[\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4]$, where $\mathcal{D}_1 = \{\{u\}, \{u, w\}, \{u, z, w\}\}$, $\mathcal{D}_2 = \{\{u\}, \{u, y\}, \{u, w\}, \{u, v, y\}, \{u, z, w\}\}$, $\mathcal{D}_3 = \{\{u, y\}, \{u, v, y\}, \{u, y, w\}, \{u, v, y, z\}, \{u, v, y, w\}, \{u, y, z, w\}, \{u, v, y, z, w\}\}$, and $\mathcal{D}_4 = \{\{u, y\}, \{u, v, y\}, \{u, y, w\}, \{u, v, y, w\}, \{u, y, z, w\}, \{u, v, y, z, w\}\}$.

3. Timely Diagnosis Engine

We now present a monitoring-based TIMELY DIAGNOSIS ENGINE for DESs, which generates efficiently a (sound and complete) set of minimal diagnoses at the reception of each newly-occurred observation without the support of a diagnoser. The algorithm makes use of the definitions given below.

Definition 1. *Let $\mathcal{D}$ be the diagnosis set of O. A diagnosis $\delta \in \mathcal{D}$ is* minimal *iff there is no other diagnosis $\delta' \in \mathcal{D}$ such that $\delta' \subset \delta$. The* candidate set *of O, $\Delta(O)$, is the set of minimal diagnoses in $\mathcal{D}$.*

Example 3. With reference to Example 2, the candidate set of $O = [brk, prt, prt, brk]$ is the singleton $\Delta(O) = \{\{u, y\}\}$, as every other diagnosis in the diagnosis set of O is a superset of $\{u, y\}$.

Definition 2. *Let $O = [o_1, o_2, \ldots]$ be a temporal observation of $\mathcal{X}$. The* temporal candidate set *of O is the sequence of candidate sets $\Delta(O_i)$, $i \geq 1$, namely: $\mathbf{\Delta}(O) = [\Delta(O_1), \Delta(O_2), \ldots]$, where O_i is the prefix of O up to the i-th observation.*

Algorithm 1: TIMELY DIAGNOSIS ENGINE

input : O, a temporal observation of $\mathcal{X}$ with initial state x_0
output: $\mathbf{\Delta}$, the temporal candidate set of O

1 $\mathfrak{I}_0 \leftarrow (x_0, \emptyset)$, $\mathcal{G} \leftarrow \{\mathfrak{I}_0\}$, $\mathcal{U} \leftarrow \lfloor \mathfrak{I}_0 \rfloor$, $\mathcal{N} \leftarrow \emptyset$, $\mathbf{\Delta} \leftarrow [\,]$
2 **repeat**
3 **repeat**
4 Pop an abduction item $\mathfrak{I} = (x, \delta)$ from $\mathcal{U}$
5 **if** $\mathfrak{I}$ *is unmarked in* $\mathcal{G}$ **then**
6 **foreach** $(x', f, o) \in \mathit{Front}(\mathfrak{I})$ **do**
7 $\delta' \leftarrow \delta \uplus f$, $\mathfrak{I}' \leftarrow (x', \delta')$
8 **if** $o = \varepsilon$ **then**
9 **if** $\mathfrak{I}' \notin \mathcal{G}$ **then**
10 **if** $(x', \bar{\delta}) \in \mathcal{G}$ *where* $\bar{\delta} \subseteq \delta'$ **then**
11 Mark $\mathfrak{I}'$
12 **else if** $x' \neq x$ **then**
13 Mark in $\mathcal{G}$ every $(x', \bar{\delta})$ where $\bar{\delta} \supset \delta'$
14 Push the abduction item $\mathfrak{I}'$ onto $\mathcal{U}$
15 Insert (the possibly marked) abduction item $\mathfrak{I}'$ into $\mathcal{G}$
16 **else if** *there is no* $(x', \bar{\delta}, o) \in \mathcal{N}$ *where* $\delta' \supseteq \bar{\delta}$ **then**
17 Insert (x', δ', o) into $\mathcal{N}$
18 Remove from $\mathcal{N}$ every $(x', \bar{\delta}, o)$ where $\bar{\delta} \supset \delta'$
19 Mark $\mathfrak{I}$ in $\mathcal{G}$
20 **until** $\mathcal{U}$ *is empty*
21 $\mathcal{G} \leftarrow \emptyset$, $\Delta \leftarrow \emptyset$
22 Let o' be the next observation in O
23 **foreach** $(x', \delta', o') \in \mathcal{N}$ **do**
24 Insert the abduction item $\mathfrak{I}' = (x', \delta')$ into $\mathcal{G}$ and push $\mathfrak{I}'$ onto $\mathcal{U}$
25 **if** *there is no diagnosis* $\bar{\delta} \in \Delta$ *where* $\bar{\delta} \subseteq \delta'$ **then**
26 Insert the diagnosis δ' into Δ
27 Remove from Δ every diagnosis δ'' where $\delta' \subset \delta''$
28 Append the diagnosis set Δ to $\mathbf{\Delta}$
29 $\mathcal{N} \leftarrow \emptyset$
30 **until** *all observations in* O *have been received.*

Example 4. With reference to the temporal diagnosis set of $O = [brk, prt, prt, brk]$ in Example 2, the temporal candidate set of O is $\mathbf{\Delta}(O) = [\{\{u\}\}, \{\{u\}\}, \{\{u, y\}\}, \{\{u, y\}\}]$, where, in contrast with the temporal *diagnosis* set, each candidate set is a singleton.

Definition 3. *An* abduction item $\mathfrak{I}$ *of a DES* $\mathcal{X}$ *is a pair* (x, δ)*, where* x *is a state of* $\mathit{Space}(\mathcal{X})$ *and* δ *is a diagnosis of a trajectory ending in* x*. The* frontier *of* $\mathfrak{I}$*,* $\mathit{Front}(\mathfrak{I})$*, is a set of triples* (x', f, o) *where* $\langle x, t, x' \rangle$ *is a transition in* $\mathit{Space}(\mathcal{X})$ *and* $(t, o, f) \in \mathit{Map}(\mathcal{X})$*.*

Example 5. With reference to $\mathit{Space}(\mathcal{W})$ in Figure 1 and $\mathit{Map}(\mathcal{W})$ in Example 2, an abduction item of $\mathcal{W}$ is $\mathfrak{I} = (4, \{u, w\})$, where $4 = ((\mathit{shorted}, \mathit{open}), \mathit{op})$ is a state of $\mathcal{W}$ and $\{u, w\}$ is the diagnosis of trajectory $[p_3, b_7, p_1]$ of $\mathcal{W}$ ending in state 4. The frontier of $\mathfrak{I}$ is the set of triples $\mathit{Front}(\mathfrak{I}) = \{(6, brk, \varepsilon), (7, brk, q)\}$.

We now present the pseudocode of the TIMELY DIAGNOSIS ENGINE algorithm (lines 1–30), which takes as input a temporal observation $O = [o_1, o_2, \ldots]$ of a DES $\mathcal{X}$ with initial state x_0, and generates as output the temporal candidate

set $\mathbf{\Delta}$ of O. The algorithm exploits three main data structures: a set $\mathcal{G}$ of abduction items *generated* already, a stack $\mathcal{U}$ of abduction items *under* processing, that is, relevant to the current observation, and a set $\mathcal{N}$ of triples (x, δ, o) (cf. the notion of *frontier* of an abduction item in Definition 3) relevant to any possible *next* observation o. In line 1, $\mathcal{G}$ and $\mathcal{U}$ are initialized with item $\mathfrak{I}_0 = (x_0, \emptyset)$. The idea is to keep generating the frontier of the items in $\mathcal{U}$ up to the next observation, based on the possible trajectories of $\mathcal{X}$, thereby possibly updating δ in each successive item. The body of the algorithm is composed of two nested loops . The outer loop (lines 2–30) is repeated until all observations in O have been received. The inner loop (lines 3–20) is repeated until stack $\mathcal{U}$ becomes empty, in which case no further new item can be generated for the current observation. At each iteration, an abduction item $\mathfrak{I} = (x, \delta)$ is popped from $\mathcal{U}$ and, if it is unmarked in $\mathcal{G}$, that is, neither processed nor pruned, each triple (x', f, o) in its frontier is considered (lines 6–18). First, an item $\mathfrak{I}' = (x', \delta')$ is generated (line 7), where $\delta' = \delta \uplus f$ is the extension of δ by f, which has no effect when $f = \varepsilon$. Then, two scenarios are considered: either when $o = \varepsilon$ (lines 8–15) or $o \neq \varepsilon$ (lines 16–18). When $o = \varepsilon$, item $\mathfrak{I}'$ is still relevant to the current observation. Thus, if not already processed (line 9), in order to avoid processing items derived from $\mathfrak{I}'$ that cannot lead to new minimal diagnoses, $\mathfrak{I}'$ is marked in case δ' is a superset of a diagnosis $\bar{\delta}$ relevant to an item sharing the same state x' in $\mathfrak{I}'$ (lines 10–11); otherwise (lines 12–13), if $x' \neq x$, every item $(x', \bar{\delta})$, where $\bar{\delta}$ is a (strict) superset of δ', is marked to avoid computing non-minimal diagnoses: in this case, $\mathfrak{I}'$ is pushed onto $\mathcal{U}$ (line 14). In either case, $\mathfrak{I}'$ (which may have been marked in line 11) is eventually inserted into $\mathcal{G}$ (line 15). When, instead, o is observable (lines 16–18), a new triple (x', δ', o) is inserted into $\mathcal{N}$ provided that there is no other triple $(x', \bar{\delta}, o)$ in $\mathcal{N}$ (that is, relevant to same state x' and same observation o) where δ' is a superset of $\bar{\delta}$ (otherwise, triple (x', δ', o) is bound to lead to a non-minimal diagnosis when processing the next observation o). Furthermore, once inserted the new triple, all the other triples $(x', \bar{\delta}, o)$ in $\mathcal{N}$, where $\bar{\delta}$ is a (strict) superset of δ', are removed for the same reasons (non-minimality). At the end of the iteration (line 19), once all triples in the frontier of $\mathfrak{I}$ have been processed (and the corresponding new items $\mathfrak{I}'$ have been generated), item $\mathfrak{I}$ is marked in $\mathcal{G}$. When $\mathcal{U}$ becomes empty, the inner loop terminates (line 20). Then, the occurrence of the next observation o' triggers the computation of the next candidate set $\Delta = \Delta([o_1, \ldots, o'])$ in lines 23–27. Specifically, $\mathcal{G}$ and $\mathcal{U}$ are initialized with items (x', δ'), where (x', δ', o') is a triple in $\mathcal{N}$ relevant to observation o'. Additionally, a diagnosis δ' involved in such items is inserted into Δ, provided that, in order to preserve minimality, Δ does not include a diagnosis $\bar{\delta}$ that is a subset of δ'. If δ' is inserted into Δ, then, still in order to preserve minimality, every diagnosis δ'' in Δ that is a superset of δ' is removed from Δ (lines 25–27). Eventually $\mathbf{\Delta}$ is extended by Δ and the set $\mathcal{N}$ of triples is emptied before the next iteration of the outer loop: this way, the inner loop will start its computation with the initial items relevant to the new current observation o'.

4. Experimental results

The timely diagnosis technique presented in this paper was prototyped in the *C* programming language and tested on a laptop with CPU Intel i7-13620H @ 4.700GHz, 32GB of RAM, under the *Linux* operating system. For comparison purposes, besides TIMELY DIAGNOSIS ENGINE, also a state-of-the-art DIAGNOSIS ENGINE was developed, which generates the *whole* diagnosis sets of O. Also, a special-purpose language for the specification of diagnosis problems was designed and developed. Both diagnosis engines were run on several diagnosis problems. Hereafter, we present a reference diagnosis problem that involves a DES with 10 components, each component model including 3 states and 6 transitions, and a mapping table that assigns a varying degree of observability and abnormality to each component based on a domain of 3 observations and 20 faults. Results are presented in Table 2, where numbers in blue boxes refer to TIMELY DIAGNOSIS ENGINE, while numbers in gray boxes refer to DIAGNOSIS ENGINE. The temporal observation consists of 8 observations. TIMELY DIAGNOSIS ENGINE was compared with DIAGNOSIS ENGINE in both the cardinality of candidate sets generated in $\mathbf{\Delta}(O)$ and the processing time for generating these sets. Specifically, for each cardinality of candidates / diagnoses, ranging in 1 .. 12, and for each observation index, ranging in 1 .. 8, the number of corresponding candidates (blue) and diagnoses (gray) are listed. For instance, at the first observation, TIMELY DIAGNOSIS ENGINE generates 5 candidates including 1 fault and 5 candidates including 2 faults, while DIAGNOSIS ENGINE generates several candidates with cardinality ranging in 1 .. 6. Total numbers of candidates / diagnoses are listed in the successive row. For instance, at the first observation, the number of candidates is 10, while the number of diagnoses is 240. Additionally, the table shows the *cardinality gain* $\mathcal{R}_\delta$, defined as the percentage of diagnoses missing in $\Delta(O_i)$ compared with the number of diagnoses in $\mathcal{D}(O_i)$, namely, $\mathcal{R}_\delta = 100 * (|\mathcal{D}(O_i)| - |\Delta(O_i)|)/|\mathcal{D}(O_i)|$.

Table 2: Compared experimental results for TIMELY DIAGNOSIS ENGINE and DIAGNOSIS ENGINE.

Cardinality of candidates / diagnoses	*Observation index*															
	1		*2*		*3*		*4*		*5*		*6*		*7*		*8*	
1	5	5	1	1												
2	5	30	1	10	1	1										
3		70	1	44	2	11	2	2								
4		80		99	3	56	3	21	2	2	2	2	1	1	1	1
5		45		118	2	157	4	94		18		18	1	11	1	11
6		10		71		253	2	241	4	68	2	66	2	49	2	49
7				17		237		376	4	161	7	150	7	133	5	131
8						123		352		235		226		220		219
9						28		183		202		201		207		207
10								41		93		93		118		118
11										19		19		38		38
12														6		6
Total candidates / diagnoses	10	240	3	360	8	866	11	1310	10	798	11	775	11	783	9	780
Cardinality gain ($\mathcal{R}_\delta$)	95.8		99.1		99.0		99.1		98.7		98.5		98.5		98.8	
Processing time (seconds)	0.02	0.18	0.17	0.66	0.89	6.61	1.63	26.36	5.53	50.96	1.02	66.94	0.37	103.19	0.39	113.25
Time gain ($\mathcal{R}_\tau$)	88.8		74.2		86.5		93.8		89.1		98.4		99.6		99.6	

For instance, the cardinality gain at the first observation is $\mathcal{R}_\delta = 95.8\%$, in other words, only 4.2% of the diagnoses generated by DIAGNOSIS ENGINE are minimal. The processing time spent by the two engines is shown in the penultimate row of the table. For instance, at the eighth (last) observation, TIMELY DIAGNOSIS ENGINE spent 0.39 seconds, while DIAGNOSIS ENGINE spent 113.25 seconds (almost two minutes). The *time gain* $\mathcal{R}_\tau$, defined as the percentage of time τ saved by TIMELY DIAGNOSIS ENGINE for generating $\Delta(O_i)$ compared with the time spent by DIAGNOSIS ENGINE for generating $\mathcal{D}(O_i)$, namely, $\mathcal{R}_\tau = 100 * (\tau(\mathcal{D}(O_i)) - \tau(\Delta(O_i)))/\tau(\mathcal{D}(O_i))$, is shown in the last row of the table. For instance, the time gain at the last observation is $\mathcal{R}_\tau = 99.6\%$.

All in all, considering that both diagnosis engines share the same data structures and auxiliary functions, TIMELY DIAGNOSIS ENGINE proved to be considerably more efficient than DIAGNOSIS ENGINE, by eliciting most likely candidates (minimal diagnoses) *without generating the complete diagnosis set upfront*. This capability is grounded on the ability of TIMELY DIAGNOSIS ENGINE to prune the search space whenever an abduction item can be ignored without the risk of jeopardizing the soundness or completeness of the resulting candidate set.

5. Related work

Model-based diagnosis of DESs can typically be employed in decision-support systems, for instance for processing alarms in remote control rooms, both for electricity [18, 19] and telecommunication networks [40]. Power transmission devices and autonomous unmanned helicopters are the focus of attention of model-based diagnosis of DESs in [35] and [18], respectively, while a latest challenge is represented by the huge realm of cyber-physical systems [12].

A recent work [30] by the authors addresses the same task as this paper, that is, *monitoring-based* diagnosis of DESs, where a new candidate set is generated at the occurrence of each observation. The major difference is that the algorithms presented in [30] compute all diagnoses whereas the diagnosis outputs considered in this paper are minimal diagnoses only. The approach in [30] is based on three different techniques: (1) *blind* diagnosis, with *no* compiled knowledge, (2) *greedy* diagnosis, with *total* knowledge compilation, and (3) *lazy* diagnosis, with *partial* knowledge compilation. By *knowledge* we mean a data structure slightly similar to a classical DES diagnoser [46], which can be generated (compiled) either entirely offline (greedy diagnosis) or incrementally online (lazy diagnosis). Experimental evidence suggests that, among these techniques for computing all diagnoses, only lazy diagnosis may be viable in non-trivial application domains. In this paper, instead, only a technique, similar to the blind one in [30], is presented. The new engine, that computes minimal diagnoses only, exhibits (experimentally recorded) processing time gains up to 99.6% with respect to the blind engine that computes all candidates.

The active system approach [32, 36], in general, and hence also the method described in this paper, represents DESs by means of complete explicit (i.e. operational) strong component models and performs abduction-based diagnosis.

More specifically, in this paper a DES is a network of communicating finite automata. There are contributions about diagnostic reasoning that ignore any explicit DES models, instead, they consider some specifications. The specification of a dynamical system (i.e. the properties the system has to exhibit over time) can be given as a formula in a temporal logic [14], such as Linear Temporal Logic (LTL) [44]. An LTL specification is the implicit representation of an automaton [9, 25, 15, 10], where LTL operators describe the state transitions. Typical diagnostic tasks are aimed at finding out whether a given behavioral evolution, called a *trace*, satisfies the specification formula and/or uncovering the causes for a trace violates the specification formula, where such causes can be searched for either in the trace [4] or in the specification [42, 43], that is, the specification may be wrong.

In this paper, a DES fault is defined as an abnormal component state transition. In the literature the notion of a fault in a DES has been generalized to the violation of a predefined behavioral property, as in [16], or to a pattern of transitions, called a *supervision pattern* [22], which can represent the occurrence of a single fault as well as of multiple faults, the ordered occurrence of significant events, the multiple occurrences of the same fault, etc. Most generally, a fault can be defined as an event that arises when a sub-trajectory of the DES matches a given pattern, like in [34, 38]. The notion of diagnosis as the set of faults on a trajectory that entails the given temporal observation applies also when the faults are generalized. Hence, minimal candidates could in principle be computed also for generalized faults.

To the best of our knowledge, the first work that adopted subset-minimal diagnoses for DESs is [49]. Differently from the proposal presented here, (*a*) the method in [49] is based on a total offline knowledge compilation, aimed at generating a so-called *minimal diagnoser*, and (*b*) in [49], computing minimal candidates only does not translate into any pruning of the search space; on the contrary, the construction of the minimal diagnoser requires the previous generation the whole DES space, which is unfeasible for real systems. In the process described in this paper, instead, no knowledge compilation is performed, every (online) observation-driven trajectory reconstruction explores just a pruned portion of the search space, and the unavailability of the DES space is assumed.

6. Conclusion

This paper proposes an algorithm, called TIMELY DIAGNOSIS ENGINE, that deals with two major difficulties of model-based diagnosis of DESs. The former difficulty is a long processing time, because of the huge size of the DES space in which the search for the trajectories complying with a temporal observation is carried out. To give an idea, if the DES includes 20 components and 20 links, with each communicating automaton comprising 4 states and each link containing one out of two different events, in the worst case, considering that a link may also be empty, the number of states in the space is $4^{20} \cdot 3^{20} = 12^{20}$, a 22-digit number. The latter difficulty is a possibly large number of diagnoses explaining the perceived sequence of observations, which may cause a cognitive overload in human diagnosticians or even delays in post-processing. These difficulties add up and become critical in scenarios where a faulty/dangerous situation should be detected as early as possible and an action should be taken in real-time. To overcome these difficulties and reduce their inherent risks, the algorithm described in this paper computes minimal diagnoses only, while performing a search in a *pruned* space: the portions of the DES space that are not relevant to minimal diagnoses are not explored. This way, as confirmed by the experimental evidence presented in this paper, a significant smaller set of diagnoses (candidates) are computed in a considerably shorter time, without however missing most likely diagnoses, thereby allowing recovery actions to be figured out and taken (ideally) in real-time.

The proposed algorithm, possibly extended with new capabilities, may be a contribution to make diagnosis of DESs viable in real-time application domains. In fact, the long-term aim of the research is to demonstrate the efficacy of the proposed method in real-world dynamic environments. To achieve this goal, several studies on the application of the TIMELY DIAGNOSIS ENGINE in domains where time constraints are crucial have to be conducted. These studies are in turn enabled by the integration of the diagnosis engine with existing systems, e.g. controllers. An extensive testing of the engine across diverse scenarios and contexts is needed.

Short-term research will instead focus on enhancing the time efficiency of the engine by exploiting knowledge-compilation techniques. Given the exponential upper bound on the size of the search space, a total compilation of the DES into a diagnoser is impractical, even when performed offline. Hence, we are envisaging a *dynamical* compilation of diagnostic knowledge, whose rationale is similar to the lazy technique in [30]. Specifically, the information computed for generating each candidate set (cf. Definition 1) may be kept in a sort of *evolving* diagnoser, which is built incrementally online, when the DES is being operated and monitored by the diagnosis task. This approach will allow

the diagnosis engine to generate immediately a candidate set when the current diagnoser state is exited by a transition marked with the newly-occurred observation, without, however, the need to build the *whole* diagnoser upfront, in contrast with common diagnoser-based approaches in the literature, including [47, 49].

Acknowledgements

We acknowledge the support of the PNRR project FAIR – Future AI Research (PE00000013), Spoke 9 – Green-aware AI, under the NRRP MUR program funded by the NextGenerationEU, specifically by the project Argumentation for Informed Decisions with Applications to Energy Consumption in Computing – AIDECC (CUP D53C24000530001).

References

[1] Abreu, R., van Gemund, A., 2009. A low-cost approximate hitting set algorithm and its application to model-based diagnosis, in: Eighth Symposium on Abstraction, Reformulation, and Approximation (SARA'09), AAAI Press. pp. 2–9.
[2] Baroni, P., Lamperti, G., Pogliano, P., Zanella, M., 1999. Diagnosis of large active systems. Artificial Intelligence 110, 135–183. doi:10.1016/S0004-3702(99)00019-3.
[3] Basile, F., 2014. Overview of fault diagnosis methods based on Petri net models, in: Proceedings of the 2014 European Control Conference, ECC 2014, pp. 2636–2642. doi:10.1109/ECC.2014.6862631.
[4] Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R., 2009. Explaining counterexamples using causality, in: Bouajjani, A., Maler, O. (Eds.), 21st International Conference on Computer-Aided Verification (CAV 2009). Springer-Verlag, Berlin, Heidelberg, Grenoble, F. volume 5643 of *Lecture Notes in Computer Science*, pp. 94–108.
[5] Brand, D., Zafiropulo, P., 1983. On communicating finite-state machines. Journal of the ACM 30, 323–342. doi:10.1145/322374.322380.
[6] Cabasino, M.P., Giua, A., Seatzu, C., 2010. Fault detection for discrete event systems using Petri nets with unobservable transitions. Automatica 46, 1531–1539.
[7] Cassandras, C., Lafortune, S., 2008. Introduction to Discrete Event Systems. second ed., Springer, New York.
[8] Cong, X., Fanti, M., Mangini, A., Li, Z., 2018. Decentralized diagnosis by Petri nets and integer linear programming. IEEE Transactions on Systems, Man, and Cybernetics: Systems 48, 1689–1700.
[9] Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M., 1991. Memory efficient algorithms for the verification of temporal properties, in: Clarke, E., Kurshanr, R. (Eds.), Second International Conference on Computer-Aided Verification (CAV 1990). Springer-Verlag, Berlin, Heidelberg, New Brunswick, NJ, USA. volume 531 of *Lecture Notes in Computer Science*, pp. 233–242.
[10] Daniele, M., Giunchiglia, F., Vardi, M., 1999. Improved automata generation for linear temporal logic, in: Halbwachs, N., Peled, D. (Eds.), 11th International Conference on Computer-Aided Verification (CAV 1999). Springer-Verlag, Berlin, Heidelberg, Trento, I. volume 1633 of *Lecture Notes in Computer Science*, pp. 249–260.
[11] Debouk, R., Lafortune, S., Teneketzis, D., 2000. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. Journal of Discrete Event Dynamic Systems: Theory and Applications 10, 33–86.
[12] Dowdeswell, B., Sinha, R., MacDonell, S., 2020. Finding faults: A scoping study of fault diagnostics for industrial cyber-physical systems. Journal of Systems and Software 168, 1–16. doi:10.1016/j.jss.2020.110638.
[13] El Fattah, Y., Dechter, R., 1995. Diagnosing tree-decomposable circuits, in: Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995), Montreal, Quebec. pp. 1742–1748.
[14] Emerson, E., 1990. Temporal and modal logic, in: van Leeuwen, J. (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. MIT Press, Cambridge, MA, USA. chapter 16, pp. 995–1072.
[15] Gerth, R., Peled, D., Vardi, M.Y., Wolper, P., 1996. Simple on-the-fly automatic verification of linear temporal logic, in: Dembiński, P., Średniawa, M. (Eds.), Proceedings of the Fifteenth IFIP International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995. Springer US, Boston, MA, pp. 3–18. URL: https://doi.org/10.1007/978-0-387-34892-6_1, doi:10.1007/978-0-387-34892-6_1.
[16] Göessler, G., Mari, T., Pencolé, Y., Travé-Massuyès, L., 2019. Towards causal explanations of property violations in discrete event systems, in: 30th International Workshop on Principles of Diagnosis (DX-19), Klagenfurt, Austria.
[17] Grastien, A., Cordier, M., Largouët, C., 2005. Incremental diagnosis of discrete-event systems, in: Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, UK. pp. 1564–1565.
[18] Grastien, A., Haslum, P., 2011. Diagnosis as planning: two case studies, in: Scheduling and Planning Applications Workshop (SPARK 2011), Freiburg, Germany. pp. 37–44.
[19] Grastien, A., Haslum, P., Thiébaux, S., 2012. Conflict-based diagnosis of discrete event systems: theory and practice, in: Thirteenth International Conference on Knowledge Representation and Reasoning (KR 2012), Association for the Advancement of Artificial Intelligence, Rome, Italy. pp. 489–499.
[20] Hamscher, W., 1991. Modeling digital circuits for troubleshooting. Artificial Intelligence 51, 223–271.
[21] Hamscher, W., Console, L., de Kleer, J. (Eds.), 1992. Readings in Model-Based Diagnosis. Morgan Kaufmann, San Mateo, CA.

[22] Jéron, T., Marchand, H., Pinchinat, S., Cordier, M., 2006. Supervision patterns in discrete event systems diagnosis, in: Workshop on Discrete Event Systems (WODES 2006), IEEE Computer Society, Ann Arbor, MI. pp. 262–268.
[23] Jiroveanu, G., Boel, R., Bordbar, B., 2008. On-line monitoring of large Petri net models under partial observation. Journal of Discrete Event Dynamic Systems 18, 323–354.
[24] Kan John, P., Grastien, A., 2008. Local consistency and junction tree for diagnosis of discrete-event systems, in: Eighteenth European Conference on Artificial Intelligence (ECAI 2008), IOS Press, Amsterdam, Patras, Greece. pp. 209–213.
[25] Kesten, Y., Manna, Z., McGuire, H., Pnueli, A., 1993. A decision algorithm for full propositional temporal logic, in: Courcoubetis, C. (Ed.), International Conference on Computer Aided Verification (CAV 1993), Elounda, Greece, June 1993, Springer, Berlin, Heidelberg. pp. 97–109.
[26] de Kleer, J., 1984. How circuits work. Artificial Intelligence 24, 205–280.
[27] de Kleer, J., 2011. Hitting set algorithms for model-based diagnosis, in: 22nd International Workshop on Principles of Diagnosis (DX 2011), Murnau, Germany. pp. 60–67.
[28] de Kleer, J., Williams, B., 1987. Diagnosing multiple faults. Artificial Intelligence 32, 97–130.
[29] Kwong, R., Yonge-Mallo, D., 2011. Fault diagnosis in discrete-event systems: incomplete models and learning. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 41, 118–130.
[30] Lamperti, G., Trerotola, S., Zanella, M., Zhao, X., 2023. Sequence-oriented diagnosis of discrete-event systems. Journal of Artificial Intelligence Research 78, 69–141. doi:10.1613/jair.1.14630.
[31] Lamperti, G., Zanella, M., 2002. Diagnosis of discrete-event systems from uncertain temporal observations. Artificial Intelligence 137, 91–163. doi:10.1016/S0004-3702(02)00123-6.
[32] Lamperti, G., Zanella, M., 2003. Diagnosis of Active Systems: Principles and Techniques. volume 741 of *Springer International Series in Engineering and Computer Science*. Springer, Dordrecht, Netherlands. doi:10.1007/978-94-017-0257-7.
[33] Lamperti, G., Zanella, M., 2006. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. Artificial Intelligence 170, 232–297. doi:10.1016/j.artint.2005.08.002.
[34] Lamperti, G., Zanella, M., 2011a. Context-sensitive diagnosis of discrete-event systems, in: Walsh, T. (Ed.), Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011), AAAI Press, Barcelona, Spain. pp. 969–975.
[35] Lamperti, G., Zanella, M., 2011b. Monitoring of active systems with stratified uncertain observations. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 41, 356–369. doi:10.1109/TSMCA.2010.2069096.
[36] Lamperti, G., Zanella, M., Zhao, X., 2018. Introduction to Diagnosis of Active Systems. Springer, Cham. doi:10.1007/978-3-319-92733-6.
[37] Lamperti, G., Zanella, M., Zhao, X., 2020. Diagnosis of deep discrete-event systems. Journal of Artificial Intelligence Research 69, 1473–1532. doi:10.1613/jair.1.12171.
[38] Lamperti, G., Zhao, X., 2014. Diagnosis of active systems by semantic patterns. IEEE Transactions on Systems, Man, and Cybernetics: Systems 44, 1028–1043. doi:10.1109/TSMC.2013.2296277.
[39] McIlraith, S., 1997. Explanatory diagnosis: conjecturing actions to explain observations, in: Eighth International Workshop on Principles of Diagnosis (DX 1997), Mont St. Michel, France.
[40] Pencolé, Y., Cordier, M., 2005. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. Artificial Intelligence 164, 121–170.
[41] Pencolé, Y., Steinbauer, G., Mühlbacher, C., Travé-Massuyès, L., 2018. Diagnosing discrete event systems using nominal models only, in: Zanella, M., Pill, I., Cimatti, A. (Eds.), 28th International Workshop on Principles of Diagnosis (DX'17), Kalpa Publications in Computing. pp. 169–183. doi:10.29007/1d2x.
[42] Pill, I., Quaritsch, T., 2013. Behavioral diagnosis of LTL specifications at operator level, in: Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013), AAAI Press. pp. 1053–1059. URL: http://dl.acm.org/citation.cfm?id=2540128.2540280.
[43] Pill, I., Wotawa, F., 2018. Automated generation of (F)LTL oracles for testing and debugging. Journal of Systems and Software 139(C), 124–141.
[44] Pnueli, A., 1977. The temporal logic of programs, in: 8th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA. pp. 46–57. URL: https://doi.org/10.1109/SFCS.1977.32, doi:10.1109/SFCS.1977.32.
[45] Reiter, R., 1987. A theory of diagnosis from first principles. Artificial Intelligence 32, 57–95.
[46] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D., 1995. Diagnosability of discrete-event systems. IEEE Transactions on Automatic Control 40, 1555–1575.
[47] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D., 1996. Failure diagnosis using discrete-event models. IEEE Transactions on Control Systems Technology 4, 105–124.
[48] Stern, R., Kalech, M., Feldman, A., Provan, G., 2012. Exploring the duality in conflict-directed model-based diagnosis, in: Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12), pp. 828–834.
[49] Zhao, X., Ouyang, D., Lamperti, G., Tong, X., 2020. Minimal diagnosis and diagnosability of discrete-event systems modeled by automata. Complexity 2020, 1–17. doi:10.1155/2020/4306261.