Using Delay Blocks to Make Non-Diagnosable Discrete Event Systems Diagnosable

Lulu He, Philippe Dague, Lina Ye

Using Delay Blocks to Make Non-Diagnosable Discrete Event Systems Diagnosable

Lulu He[1], **Philippe Dague**[1] and **Lina Ye**[1,2]
[1]Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles
[2]CentraleSupélec
e-mail: firstname.name@universite-paris-saclay.fr

Abstract

Fault diagnosis is a crucial and challenging task in the automatic control of complex systems, whose efficiency depends on a system property called diagnosability. Diagnosability describes the capability of a system to determine with certainty whether a fault has effectively occurred, based on a sequence of observations. The diagnosability problem of discrete event systems has received considerable attention in literature. However, what if a system is revealed as non-diagnosable? One classical way is to make more events observable by adding new sensors. In this paper, we propose a new non-intrusive way to make a non-diagnosable system diagnosable by merely adding delay blocks on some observable events, thus deferring their observations. As far as we know, this is the first attempt to remove non-diagnosability with delay blocks without using controllable events or changing the structure of the system. Our approach is encoded as an SMT formula, whose efficiency is demonstrated by our experimental results.

1 Introduction

Fault diagnosis is a crucial and challenging task in the automatic control of complex systems ([1; 2; 3; 4; 5; 6; 7; 8]), whose efficiency depends on a system property called diagnosability. The diagnosability problem for discrete event systems (DESs) has received considerable attention in literature. Diagnosability describes the system's ability to determine whether a fault has effectively occurred based on the observations. In a given DES, the existence of two infinite behaviors, with the same observations but one containing the considered fault and not the other, called a critical pair, is a witness of non-diagnosability. The existing work searches for such critical pairs both in centralized ([9; 10; 11; 12; 13]) and distributed ([14; 15; 16]) ways. The most classical method is to construct a structure called twin plant that captures all pairs of observationally equivalent behaviors to directly check the existence of critical pairs.

As diagnosability is one critical property of the system, it is important to design a diagnosable system, which will prove to be substantially convenient for subsequent system diagnosis. But this goal is not trivial: diagnosability is a quite strong property, thus designed systems are very often non-diagnosable. Up to now, most work has focused on checking diagnosability of the system, while little work has considered improving diagnosability efficiently and economically. It is thus interesting to study how to transform non-diagnosable systems into diagnosable ones. One simple way is to add sensors to increase the observability of the system. Another way relies on introducing control in order to constrain the system's behaviors in such a way that the allowed behaviors are diagnosable. A last way is to redesign a new system, which brings back the issue of guaranteeing its diagnosability. All these means are costly and intrusive.

In this paper, we propose a new non-intrusive (without changing the system's behaviors and without adding sensors) approach for this purpose by merely deferring some observable events, while keeping the original system structure. For the sake of simplicity, this work has been conducted on classical DESs, modeled by finite state automata (FSA). We calculate a minimal set of observable events, whose deferral can make a non-diagnosable system into a diagnosable one, as a min-cut of the set of observable transitions in the normal path (and not common to the faulty path) of the unfolded critical pairs. We make use of the max-flow min-cut theorem to compute by advance the minimal size of a cut. More precisely, we adapt finite automata with time-delay blocks (ADB) ([17]) to eliminate all existing critical pairs witnessing non-diagnosability by taking care not to create new ones, so that every faulty trajectory can be distinguished by observation from all normal ones. This computation is encoded in Satisfiability Modulo Theories (SMT), an extended form of Boolean satisfiability (SAT), where literals are interpreted w.r.t. a background theory. The reason that we chose SMT instead of SAT is to make our approach extensible in order to handle timed automata in a future work.

Our contribution to the design of diagnosable DESs is multifold. First, in order to eliminate efficiently all pairs violating diagnosability by adding the fewest delay blocks possible, we compute a minimal-size set of transitions in normal trajectories according to the max-flow min-cut theorem, encoded in SMT. Secondly, we analyze the scope of our approach by characterizing the systems for which it is applicable. Thirdly, we present experimental results on benchmarks to demonstrate the efficiency of our approach.

2 Preliminaries

In this section, after a reminder of the diagnosability for DESs and the introduction of ADBs, we present how to use the latter to make a non-diagnosable system diagnosable.

2.1 Diagnosability of DESs

Definition 1. (System Model) A DES is modeled as an FSA, denoted by $G = (Q, \Sigma, \delta, q_0)$, where:

- Q is a finite set of states;
- Σ is a finite set of events;
- $\delta \subseteq Q \times \Sigma \times Q$ is a finite set of transitions;
- $q_0 \in Q$ is the initial state.

The set of events Σ is divided into three disjoint parts: $\Sigma = \Sigma_o \uplus \Sigma_u \uplus \Sigma_f$, where Σ_o is the set of observable events, Σ_u the set of unobservable normal events and Σ_f the set of unobservable fault events. We extend $\delta \subseteq Q \times \Sigma \times Q$ to $\delta \subseteq Q \times \Sigma^\star \times Q$ in a straightforward way.

Consider the simple system model depicted in Figure 1, where $\Sigma_o = \{o1, o2, o3\}$, $\Sigma_u = \{u\}$, $\Sigma_f = \{F\}$, and q_0 is the initial state.

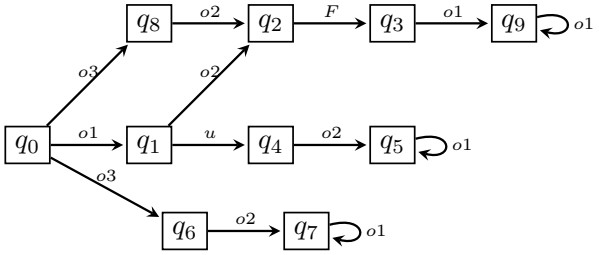

Figure 1: A system example modeled by an FSA.

Given a system model G, its prefix-closed language $L(G) \subseteq \Sigma^\star$, which describes the normal and faulty behaviors of the system, is the set of words produced by G: $L(G) = \{s \in \Sigma^\star \mid \exists q \in Q, (q_0, s, q) \in \delta\}$. In the following, we call a word from $L(G)$ a *trajectory* in G and a sequence $q_0\sigma_0q_1\sigma_1...$, with $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all i, a *path* or *run* in G, whose associated trajectory is $\sigma_0\sigma_1...$. (path and trajectory are defined identically starting from any *reachable* state, i.e., any state reached from q_0 by a path). A path $q_i\sigma_iq_{i+1}\sigma_{i+1}...\sigma_mq_i$ is called a *cycle*. Given $s \in L(G)$, we denote the post-language of $L(G)$ after s by $L(G)/s$, formally defined as: $L(G)/s = \{t \in \Sigma^\star \mid s.t \in L(G)\}$. The projection of the trajectory s to observable events is denoted by $P(s)$. These notions extend to infinite paths and trajectories.

For the sake of simplicity, we adopt the two following conventional assumptions.

Assumption 1: (Aliveness) From any state, there is at least one outgoing transition. The corresponding language $L(G)$ is thus live.

Assumption 2: (Convergence) There is no cycle containing only unobservable events.

Intuitively, a predefined fault is diagnosable in a system G if one can be sure of its occurrence after sufficient observations from G, whose formal definition is given as follows ([9]), where s^F denotes a trajectory s ending with fault F.

Definition 2. (Diagnosability) A fault $F \in \Sigma_f$ is $diagnosable$ in a DES model G iff

$$\exists k \in N, \forall s^F \in L(G), \forall t \in L(G)/s^F, (|t| \geq k \Rightarrow \forall p \in L(G), (P(p) = P(s^F.t) \Rightarrow F \in p)).$$

The above definition states that for each trajectory s^F in G, for each t that is an extension of s^F with sufficient events, every trajectory p in G that is observationally equivalent to $s^F.t$ should contain F. In other words, the diagnosability checking consists in verifying the non-existence of a pair of trajectories p and p' satisfying the following three conditions: 1) p contains F and p' does not; 2) p and p' are infinite; 3) $P(p) = P(p')$. Such a pair is called a $critical\ pair$ ([12]), which has been proven to violate diagnosability defined by Definition 2 and thus witnesses non-diagnosability. In the example of Figure 1,there are two critical pairs: $CP_1 = \{o1.o2.F.o1^\omega, o1.u.o2.o1^\omega\}$ and $CP_2 = \{o3.o2.F.o1^\omega, o3.o2.o1^\omega\}$. That is to say, when we observe the sequence of events $o1.o2.o1^\omega$ or $o3.o2.o1^\omega$, we can never be sure of the occurrence of the fault as two trajectories exist for which only one contains the fault, while both have exactly this same sequence as observations.

Theorem 1. *Given a system model G, a fault F in G is diagnosable iff there is no critical pair w.r.t. F in G.*

Classically, diagnosability is checked for one single fault F (with multiple occurrences) at a time (the other fault being thus considered as unobservable normal events), i.e. $\Sigma_f = \{F\}$, and the approach is applied as many times as the number of faults. Our algorithm will proceed in the same way. From theorem 1, it is clear that making a non-diagnosable system model diagnosable boils down to find a way to eliminate all critical pairs without creating new ones. We explore achieving this by just deferring some observable events, i.e., acting only on observation transmission.

2.2 Twin plant for diagnosability checking

One classical algorithm to check the existence of critical pairs is to construct a structure, often called twin plant in the literature, by synchronizing normal and faulty trajectories based on observable events.

Definition 3. (Diagnoser). The diagnoser of a system model G with a considered fault F is the automaton $D_G = (Q_D, \Sigma, \delta_D, q_D^0)$, where: 1) $Q_D \subseteq Q \times \{N, F\}$ is the set of states; 2) Σ is the set of events of G; 3) $\delta_D \subseteq Q_D \times \Sigma \times Q_D$ is the set of transitions; 4) $q_D^0 = (q_0, N)$ is the initial state. The transitions of δ_D are those $((q, \ell), e, (q', \ell'))$, with (q, ℓ) reachable from q_D^0, such that there is a transition $(q, e, q') \in \delta$, and $\ell' = F$ if $\ell = F \vee e = F$, otherwise $\ell' = N$.

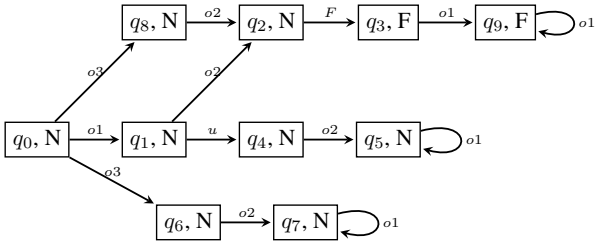

Figure 2: Diagnoser of the system in Figure 1.

Intuitively, each state of a diagnoser is a pair of a system state and a fault label such that the label is F when a fault occurs on the path being considered from the initial to the present state, N otherwise. Thus a state is split into two states in the diagnoser if it is reachable in the system by a path with F and by a path without F. Figure 2 shows the diagnoser of the system depicted in Figure 1.

Next we refine the diagnoser by keeping only observable information, which requires the following operation.

Definition 4. (ϵ-Closure). Given a FSA $G = (Q, \Sigma, \delta, q_0)$, its ϵ-*closure* w.r.t. Σ_d, with $\Sigma_d \subseteq \Sigma$, is $\mathsf{C}_{\Sigma_d}(G) = (Q_d, \Sigma_d, \delta_d, q_0)$, where: 1) $Q_d = \{q_0\} \cup \{q \in Q \mid \exists s \in \Sigma^*, \exists \sigma \in \Sigma_d, (q_0, s\sigma, q) \in \delta\}$; 2) $(q, \sigma, q') \in \delta_d$ if $\sigma \in \Sigma_d$ and $\exists s \in (\Sigma \backslash \Sigma_d)^*, (q, s\sigma, q') \in \delta$.

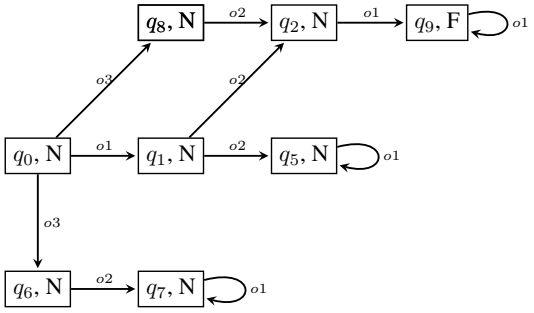

Figure 3: Refined diagnoser of the system in Figure 1.

The refined diagnoser is defined as the ϵ-closure of the diagnoser w.r.t. the set of observable event Σ_o and denoted by $D_G^r = \mathsf{C}_{\Sigma_o}(D_G)$. Figure 3 shows the refined diagnoser constructed from the diagnoser depicted in Figure 2.

Definition 5. (Twin plant). Given a system model G, its twin plant T_G is obtained by synchronizing the corresponding refined diagnoser D_G^r with itself based on the set of observable events, i.e., $T_G = D_G^r \parallel_{\Sigma_o} D_G^r$.

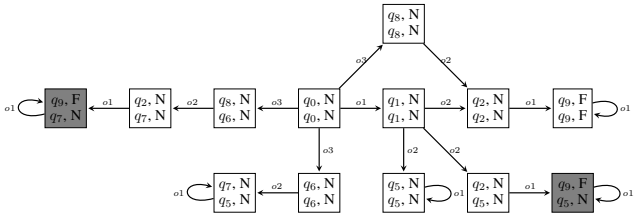

Figure 4: Twin plant of the system in Figure 1.

Note that, when starting from refined diagnosers, the synchronised product on observable events is just the product. Each state of a twin plant is composed of a pair of diagnoser states, which is called an ambiguous state if one of the fault labels is F and the other is N. An ambiguous cycle is one that contains only ambiguous states. Figure 4 depicts the twin plant based on the refined diagnoser shown in Figure 3. Note that the gray nodes are ambiguous states, whose self-cycles are thus ambiguous cycles.

As our goal is to identify all critical pairs in order to eliminate them later by adding delay blocks, we can further reduce the twin plant by retaining only its parts that are useful for this purpose.

Definition 6. (Critical twin plant). Given a twin plant T_G, its critical version T_G^c is obtained by keeping only all its states from which an ambiguous cycle is reachable (and removing possible normal cycles that could remain on paths from the initial state to such an ambiguous cycle).

The critical twin plant T_G^c obtained from the twin plant T_G of Figure 4 is shown in Figure 5. Actually the label N, which is not necessary for our next analysis, can be omitted.

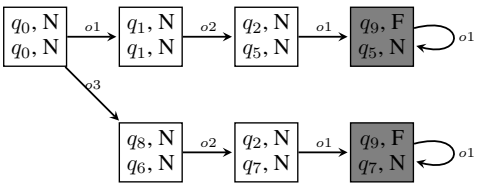

Figure 5: Critical twin plant of the system in Figure 1.

2.3 Automata with delay blocks (ADB)

We now introduce finite automata with delay blocks (ADB) ([17]) by adapting them to our problem. ADB, obtained by extending finite state automata with time-delay blocks, can be used to defer some observable events. Our idea is to use the deferral aspect of ADB to eliminate all critical pairs in the twin plant without creating new ones, so that every faulty trajectory can be distinguished from normal ones. Values (in time units) of delays can be in general any natural integer but we need only 0 and 1 delays.

Definition 7. (ADB) An automaton with delay blocks (ADB) is a quintuple $\mathcal{B} = (Q, D, \Sigma, \delta, q_0)$, where:

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $\Sigma = \Sigma_o \uplus \Sigma_u \uplus \Sigma_f$ is a finite set of events;
- $D = \{d_0, d_1\}$ is the set of delay blocks, with respective delays 0 and 1, where the value 1 is used to defer the corresponding observable event by one time unit;
- $\delta \subseteq Q \times ((\Sigma_o \times D) \uplus (\Sigma_u \uplus \Sigma_f)) \times Q$ is a finite set of transitions.

There are two kinds of transitions according to the types of events:

- $(q, \sigma, d, q') \in \delta$, where $\sigma \in \Sigma_o$, denoting a transition from q to q', with the event σ whose observation is either deferred when the associated value of the delay block d is 1, or not otherwise.
- $(q, \sigma, q') \in \delta$, where $\sigma \in \Sigma_u \uplus \Sigma_f$, unobserved transition without delay block.

For a finite word w, let $|w|$ denote its length, and $w[i]$ its $(i+1)$th element (starting from 0), if $|w| > i$. Given an ADB $\mathcal{B}$, its timed language $L(\mathcal{B})$ is defined by:

$$L(\mathcal{B}) = \{w \in (\Sigma \times \{0,1\})^* \mid \exists q \in Q, (q_0, w, q) \in \delta\}$$

where the timed word w is a finite string of tuples $\langle \sigma, d \rangle \in (\Sigma_o \times \{0,1\}) \uplus ((\Sigma_u \uplus \Sigma_f) \times \{0\})$, all unobservable events being implicitly associated with 0. We refer to the first component of the tuple $w[i]$ as the $event$ and to the second as the $timestamp$, i.e., either 0 (not deferred) or 1 (deferred).

We redefine for an ADB the projection operator P of a finite timed trajectory w on observable events. Given w, $P(w) = \{P_0(w), P_1(w)\}$ is made up of two sequences of observable events in w whose timestamps are all 0 or 1, respectively. Given w and w', $P(w) = P(w')$ iff $P_0(w) = P_0(w')$ and $P_1(w) = P_1(w')$. We will write in short $P(w)$ as $P_0(w)\#P_1(w)$ where $\#$ denotes the demarcation between time slot 0 and time slot 1.

Consider the example in Figure 6, which is an ADB extended from the FSA of Figure 1. Here, the delay values for all observable transitions are 0 and omitted, except for the transitions $(q_4, o2, q_5)$ and $(q_0, o3, q_6)$, where

the value is 1. In this ADB, the timed trajectories corresponding to CP_1 are $\langle o1,0\rangle\langle o2,0\rangle\langle F,0\rangle\langle o1,0\rangle^\omega$ and $\langle o1,0\rangle\langle u,0\rangle\langle o2,1\rangle\langle o1,0\rangle^\omega$, whose projections on observable events are $o1o2o1^\omega\#$ and $o1^\omega\#o2$ respectively, thus distinct. And the timed trajectories corresponding to CP_2 are $\langle o3,0\rangle\langle o2,0\rangle\langle F,0\rangle\langle o1,0\rangle^\omega$ and $\langle o3,1\rangle\langle o2,0\rangle\langle o1,0\rangle^\omega$, whose projections on observable events are $o3o2o1^\omega\#$ and $o2o1^\omega\#o3$ respectively, thus distinct. So, thanks to the delay blocks, F becomes diagnosable.

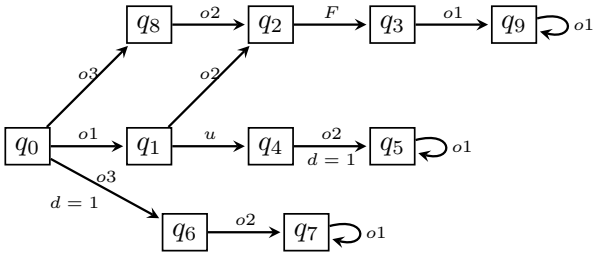

Figure 6: An example of ADB from the system in Figure 1.

Note that the introduction of delay blocks changes in general the order of observable events in a run. But it may happen the order remains unchanged and nevertheless the observations can be distinguished thanks to the timestamps that are assumed to be observed too (equivalent to $\#$ being observable). For example, if the second 1-valued delay block is with the transition $(q_7, o1, q_7)$ instead of $(q_0, o3, q_6)$, the observations of the timed trajectories corresponding to CP_2 are $o3o2o1^\omega\#$ and $o3o2\#o1^\omega$ respectively, so are distinct, though the ordered sequences of events are the same, i.e., $o3o2o1^\omega$. Unlike $P(L(G))$, the timed observation language $P(L(\mathcal{B}))$ is no longer live in general ($o1^n\#o2$ is not the prefix of $o1^{n+1}\#o2$), but at least one of $P_0(L(\mathcal{B}))$ or $P_1(L(\mathcal{B}))$ is live. Note also that the time chosen for the delay (here, one unit) plays actually no role: more than a time-delay, it is actually a logical delay. But time-delays could be useful if we would consider time automata instead of automata. As for material implementation in a physical device, one can imagine to send the observable events issued from the sensors equipped in the model with a 1 delay block to a different terminal than the default one used for communication from all other sensors, being assumed that the order of observable events in a run is respected by the communication channel to each display. When simply changing the order of observable events is enough, implementing time-delay blocks could be done simply by buffering the signal issued from the concerned sensors.

3 Computing delay blocks for making a non-diagnosable system diagnosable

Our novel diagnosable transformation approach, based on ADBs, which delays some observations, comprises the following major steps:

1. Synchronizing on observable events the refined diagnoser D_G^r with itself to get the twin plant T_G, as seen above.
2. Unfolding its critical version T_G^c into a flow network FN.
3. Calculating a constrained min-cut set of FN and adding to each selected transition of this min-cut a 1-valued delay block such that the corresponding critical pairs disappear in the resulting ADB.

3.1 Unfolding twin plant into flow network

We now briefly introduce the classical notion of flow network before showing how to unfold a given FSA into a specific flow network adapted to our approach (see [18] for more details about flow network theory).

Definition 8. (Flow network) A flow network is a directed and connected graph $FN = (V, E)$ with a capacity function w assigning a nonnegative weight to each edge:

- V is the set of nodes (vertices);
- $E \subseteq V \times V$ is the set of edges;
- source node $s \in V$ and target node $t \in V$ are two distinguished nodes;
- w is a function : $E \to \mathbb{R}_\infty = \mathbb{R}_+ \cup \{+\infty\}$.

Given an FSA, we unfold it as a flow network, by adapting the Floyd's cycle detection algorithm in order to link the last node of each cycle to a newly created node, the target node t. We just extend slightly the definition of a flow network by keeping the original transition labels of the FSA in addition to the weights. And, in our case, we give each edge the unique weight 1 since our goal is to find a min-cut set of all the critical pairs.

We thus unfold T_G^c as a flow network FN. The way we construct FN guarantees that any reachable cycle in T_G^c is transformed into a path reaching the target node t (note that T_G^c contains only ambiguous cycles). In other words, if we can block (role of the min-cut set) all paths from s to t, we can then forbid (by adding delay blocks to transitions present in the min-cut set) all ambiguous cycles, thus all critical pairs.

Figure 7 shows the unfolding FN of the critical twin-plant T_G^c of Figure 5.

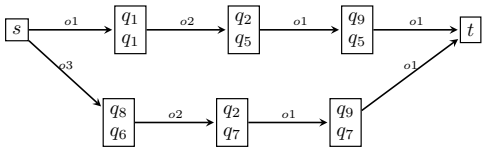

Figure 7: Flow network obtained by unfolding the critical twin plant of the system in Figure 1 (all capacities equal 1).

3.2 Encoding min-cut

To reduce costs (of installing delay blocks), we want to calculate a minimum number of transitions, such that adding delay blocks to them will eliminate all existing critical pairs simultaneously. We encode this in SMT as a min-cut problem. In graph theory, a cut of a given flow network partitions the set of nodes into two disjoint subsets.

Definition 9. (Min-cut) Given a flow network $FN = (V, E)$ with the source node s and the target node t, an s-t cut (S, T) is a partition of V such that $s \in S$ and $t \in T$. The corresponding set of cutting edges $X_{(S,T)}$ is:

$$X_{(S,T)} := (S \times T) \cap E = \{(u, v) \in E \mid u \in S, v \in T\}.$$

The capacity $c(S, T)$ of a cut (S, T) is the total weight of the corresponding cutting edges:

$$c(S,T) := \Sigma_{(u,v)\in X_{(S,T)}}\, w(u,v).$$

A min-cut of FN is a cut whose capacity is minimum over all cuts of FN.

For an s-t cut, if we remove all its cutting edges, no path exists from s to t and consequently there is no positive flow from s to t. Conversely, finding a subset of edges whose removal interrupts the flow from s to t defines an s-t cut.

Now we show how to build an SMT formula Ψ, whose any satisfying assignment corresponds to a cut set. Ψ is composed of different parts to provide a comprehensive description. Given a flow network FN obtained by unfolding an FSA as above, we first encode the essential static parts and then the S and T sets by integers 0 and 1 through a function $C : V \to \{0,1\}$, where integer 0 encodes the nodes in S and 1 the nodes in T.

Initialization

The source state s and the target state t should be in S and T, respectively.

$$\Phi^{Init} := ((C(s) = 0) \wedge (C(t) = 1))$$

Uniqueness

We formalize here that every state is in S or in T. In addition, due to the way an SMT codes a function, a variable cannot be assigned different values simultaneously, that is, a state cannot both belong to S and T.

$$\Phi^{Uniqueness} := (\bigwedge_{q\in V} (C(q) = 0 \vee C(q) = 1))$$

Weight of the edges

The weights of the edges are valued by 1 or 0 through a function $CE : E \to \{0,1\}$, according to their membership to the cut set or not (remember that capacities of all edges are equal to 1). For any edge $(q,q') \in E$, with $q \in S$ and $q' \in T$, this edge is in the cut set and its weight is 1. Otherwise, its weight is 0.

$$\Phi^{Weight} := (\bigwedge_{(q,q')\in E} (C(q) = 0 \wedge C(q') = 1 \Rightarrow CE(q,q') = 1) \wedge (C(q) = 1 \vee C(q') = 0 \Rightarrow CE(q,q') = 0))$$

Size of the cut

In the calculation process, at each step i, we consider a new edge (q,q'), whose weight is added to the current flow.

$$\Phi^{Update} := ((\bigwedge_{i=1}^{n-1} (process(i) = (process(i-1) + CE(q,q')))) \wedge (process(0) = 0))$$

where n is the number of edges. Here the function $process(i)$ denotes the current number of edges in the cut set in construction at step i, which is updated based on the weight of the current edge $(q,q') \in E$ chosen at step i and the previous number. This function is initialized to 0.

Bounding the size

B being a given upper-bound of the cut's size, we are only interested in computing a cut of size at most B.

$$\Phi_B^{Bound} := (\bigwedge_{i=0}^{n-1} (process(i) \leq B))$$

We have formalized all conditions, whose satisfying assignment can be easily proved to correspond to a cut set of size at most B. Thus we have:

$\Psi_B := \Phi^{Init} \wedge \Phi^{Uniqueness} \wedge \Phi^{Weight} \wedge \Phi^{Update} \wedge \Phi_B^{Bound}$

and the cut set corresponding to any given solution is obtained as the set of the edges (q,q') such that $CE(q,q') = 1$.

Min-cut

To compute a min-cut set, we have just to iterate successive calls to the SMT solver about satisfiability of Ψ_B, with B initialized to $+\infty$ and updated before each new call to $B := M - 1$, where M is the value of the size of the cut set solution of the last call, up to an unsat answer. Then the solution of the last successful call is a min-cut set.

SMT formula for a min-cut

Now, we can improve the computation of a min-cut by avoiding this iteration process thanks to the use of the max-flow min-cut theorem and of the Ford-Fulkerson algorithm.

Theorem 2. *(Max-flow min-cut theorem) In a flow network, the value of a maximum flow passing from the source node to the target node is equal to the capacity of a min-cut.*

The computation of a min-cut set is thus as follows:

(i) Pre-computing the value MF of a max-flow based on the Ford-Fulkerson algorithm. MF is equal to the capacity of a min-cut by the max-flow min-cut theorem and thus equal to the number of all the cutting edges.

(ii) Calculating the cutting edges of a min-cut through SMT encoding as above by using this value MF as bound B.

$\Psi := \Phi^{Init} \wedge \Phi^{Uniqueness} \wedge \Phi^{Weight} \wedge \Phi^{Update} \wedge \Phi_{MF}^{Bound}$

Note that a min-cut set is not unique, but the formula Ψ ensures that for every path from s to t, representing a critical pair, at least one of its transitions will be counted in the min-cut set. For the unfolded T_G^c of the Figure 7, there are a priori sixteen (4×4) min-cut sets of size two (i.e., the value of the max-flow or min-cut), corresponding to its eight $(4 + 4)$ transitions.

3.3 Computing delay blocks

Applied to the flow network FN which is the unfolding of the critical twin plant T_G^c of the system G, a min-cut set is thus made up of pairs of transitions in G, each one corresponding to observable transitions in the faulty path and the normal path, respectively, of some critical pair. The idea is thus to defer the observable event in one or the other transition of such each pair by adding to it a 1-valued delay block. But it may happen that both transitions of a pair are equal, what we will call a twin-transition. This is the case of $(q_0, o1, q_1)$ from the example depicted in Figure 1, which appears twice in the transition $(((q_0,N),(q_0,N)), o1, ((q_1,N),(q_1,N)))$ of the corresponding T_G^c of Figure 5 and thus in the first transition $(s, o1, ((q_1,N),(q_1,N)))$ of its unfolded FN of Figure 7. Obviously, adding a delay block to a twin-transition would have the same deferring effect on the normal and faulty trajectories of the concerned critical pair, that will thus not be disambiguated. So, all min-cut sets containing a twin-transition have to be discarded. For the example, it means that min-cut sets containing the first transition of FN above have to be discarded and thus only twelve (3×4) min-cut sets remain. In practice, to avoid filtering the min-cut sets afterwards, we use one or the other of the following equivalent two methods.

(A) Eliminate the twin-transitions when constructing the unfolding of T_G^c by considering them as silent and applying the ϵ-$closure$ (see Figure 8 (above)).

(B) Assign the weights of these twin-transitions to $+\infty$ and consider only those min-cut solutions whose capacity is a finite integer (see Figure 8 (below)).

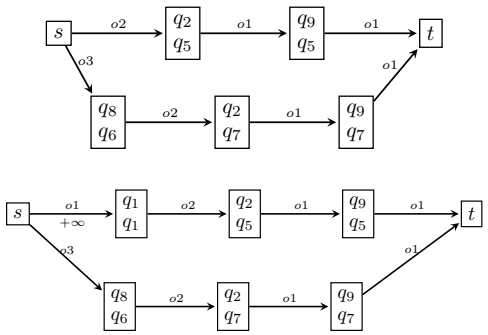

Figure 8: Flow network of Figure 7, where common transitions are eliminated (above) or assigned infinite capacity (below).

Now, it is reasonable, in a pair of transitions of G constituting an element of a min-cut set, to choose to add a 1-valued delay block to the transition belonging to the normal path (as the normal part of G is the one issued from its nominal design). Thus a solution to our attempt to make the system G diagnosable will be obtained from any given min-cut set X by adding a 1-valued delay block to all the transitions in G appearing in X as the normal components (in our case, the second component) of the pairs of transitions it is made up of. We denote by δ_X the set of these transitions:

$$\delta_X = \{t \in \delta \mid \exists t_F \in \delta, \exists t', t'' \in \delta^* \ (t''t_F, t't) \in X\}$$
where $X = \{e \in E \mid CE(e) = 1\}$ in a solution of Ψ

(use of t' and t'' is due to the refinement process applied). And we know by the elimination above that $t_F \neq t$.

Note that it may happen that two distinct transitions of the min-cut set X have the same normal component (this means that this normal transition appears in two different critical pairs). Thus $|\delta_X|$ may be smaller than $|X|$. It may also happen that two different min-cut sets X and X' give rise to the same solution in terms of transitions to be delayed: $\delta_X = \delta_{X'}$. Actually, though by construction there is no cycle in FN, which is derived from the unfolded T_G^C, it may occur cycles, and thus several occurrences of a same transition of G, in the normal path corresponding to a path from s to t in FN, thus a same transition of G may appear several times as a candidate to block some critical pair. This is the case of the example as the two transitions $((q_2, q_5), o1, (q_9, q_5))$ and $((q_9, q_5), o1, t)$ of Figure 8 give rise to the same normal transition $(q_5, o1, q_5)$ of G (see Figure 1) to delay, and the two transitions $((q_2, q_7), o1, (q_9, q_7))$ and $((q_9, q_7), o1, t)$ give rise to the same normal transition $(q_7, o1, q_7)$ of G to delay. There are thus only six ways of adding two 1-valued delay blocks to transitions of G to make G diagnosable:
$\{(q_4, o2, q_5), (q_0, o3, q_6)\}$, $\{(q_4, o2, q_5), (q_6, o2, q_7)\}$,
$\{(q_4, o2, q_5), (q_7, o1, q_7)\}$, $\{(q_5, o1, q_5), (q_0, o3, q_6)\}$,
$\{(q_5, o1, q_5), (q_6, o2, q_7)\}$, $\{(q_5, o1, q_5), (q_7, o1, q_7)\}$.
The first solution corresponds to the ADB of Figure 6. Solving Ψ provides one of these solutions and all six can be obtained by enumerating all solutions. Computing the max-flow and computing a min-cut have polynomial complexity, as all constructions above (such as the twin plant), thus the whole method has a polynomial complexity. It is also polynomial in the number of distinct faults, as each fault is processed at its turn.

3.4 Diagnosability making condition

If the method above finds a solution X (a finite min-cut set) after having dealt with twin-transitions by the method (A) or (B) above, it is thus clear from the construction that adding 1-valued delay blocks to all transitions in δ_X eliminates all critical pairs in the system G. But the method can fail to provide a solution due to the existence of twin-transitions. If the method (A) is used to get rid of these twin-transitions, the method fails if and only if there is a path in FN from s to t composed only of twin-transitions, giving thus rise after their elimination to an empty path from s to t for which no cut exists. If it is the method (B) which is used, then the method fails if and only if the pre-computed value of the max-flow equals $+\infty$. Which means again that it exists a path in FN from s to t composed only of twin-transitions (of weight $+\infty$), the only way to get an infinite flow as all non-twin-transitions have weight 1. In this case, it is obvious that is is impossible to eliminate all critical pairs by adding delay blocks.

Now, does the elimination of all critical pairs, when possible, ensure the diagnosability of G equipped with the delay blocks? Unfortunately, not in general, because the newly created that way sequences of observations (i.e., containing at least one deferred observation) may create new critical pairs. But note that this can occur only in very special cases: necessarily the faulty path of such a critical pair has to contain a delay block, thus a transition belonging to a normal path of some critical pair. This is the case of the system depicted at left of Figure 9: in the only critical pair made up of the normal path $t2t3t6^\omega$ and the faulty path $t2t4t5t6^\omega$, $t2$ and $t6$ are twin-transitions and thus the only way to eliminate this critical pair is to add a 1-valued delay block to transition $t3$, which also belongs to the faulty path $t1t3t6^\omega$. And actually the obtained system is not diagnosable as this faulty path constitutes a new critical pair with the normal path $t2t3t6^\omega$, both having now the same observation $o1^\omega \# o2$. But note that we can iterate our method from this ADB and this new critical pair: $t3$ and $t6$ are now twin-transitions and the only solution is to add a 1-valued delay block to transition $t2$. Now, $t2t3t6^\omega$ has for observation $o1^\omega \# o1o2$, distinguishable from $o1^\omega \# o2$. And it is easily verified that no new critical pair is created.

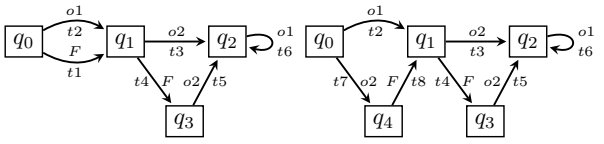

Figure 9: Two systems with twin-transitions $\{t2, t6\}$ and common transitions $\{t2, t6, t3\}$, that do not satisfy the sufficient condition for diagnosability. Adding a 1-valued delay block to transition $t3$ is the only way to eliminate the only critical pair, but the obtained system at left is still non-diagnosable while the one at right is now diagnosable.

Thus a sufficient condition to avoid creating new critical pairs and ensuring diagnosability is to forbid a delay block to take place at a transition that belongs to both an infinite normal path and an infinite faulty path, i.e., to both the normal and the faulty diagnoser, what we will call a common transition (e.g., $t3$ in our example). So, we are sure that a given solution X ensures diagnosability if X does not contain any common transition. Note that we can get rid of

	$\|Transitions\|$	$\|Clauses\|$	$\|Cut\|$	$SAT?$	$Time\ (without\ MF,\ with\ (A))$	$Time\ (with\ MF\ and\ (A))$	$Time\ (with\ MF\ and\ (B))$
ex_1	13	79	2	Yes	10.03	7.53	8.58
ex_2	213	1703	2	Yes	11.56	7.56	8.45
ex_3	500	19782	2	Yes	80.23	20.44	22.34
$hvac_1$	14	191	2	Yes	0.72	0.58	0.64
$hvac_2$	114	1920	2	Yes	0.82	0.63	0.71
$hvac_3$	500	8576	2	Yes	40.33	10.44	11.05
Su_1	14	64	1	Yes	0.27	0.18	0.18
Su_2	108	643	1	Yes	0.50	0.61	0.66
Su_3	500	6753	1	Yes	18.22	9.83	10.30
$Mehdi_1$	12	407	2	Yes	0.70	0.18	0.32
$Mehdi_2$	116	3248	2	Yes	0.80	0.65	0.82
$Mehdi_3$	500	8975	2	Yes	49.33	12.22	13.22
$Jiang_1$	10	36	2	No	0.48	0.03	0.01
$Jiang_2$	114	450	2	No	0.63	0.42	0.01
$Jiang_3$	500	8675	2	No	28.33	9.22	4.33

Table 1: Experimental results

all common transitions in exactly the same way we did for twin-transitions (and actually twin-transitions are common transitions). Thus if a finite min-cut set is obtained after this processing, we are sure the obtained system is diagnosable. But this condition is not necessary, as shown by the system depicted at right of Figure 9, which does not satisfy it and nevertheless becomes diagnosable when a 1-valued delay block is added to $t3$.

Theorem 3. *Given a non-diagnosable system model G, and the flow network FN derived from its unfolded critical twin plant T_G^c after getting rid of twin-transitions (i.e., identical in the normal and faulty paths) either by eliminating them (A) or assigning them an infinite capacity (B), all existing critical pairs of G can be suppressed by introducing 1-valued delay blocks if and only if there does not exist an empty path from s to t (A) or, equivalently, the value MF of the max-flow of FN is finite (B). In this case the observable normal transitions to equip with delay blocks are given by δ_X for any min-cut solution X of the SMT formula Ψ. In total, finding a solution δ_X or proving it does not exist is polynomial in the input G. Now, a sufficient, but not necessary, condition for such a solution δ_X to not create new critical pairs, and thus to ensure diagnosability of the system obtained, is that δ_X does not contain any common transition, i.e., belonging to both the normal and the faulty diagnoser. The existence of at least one such solution is thus guaranteed when a solution exists after getting rid of common transitions, which can be polynomially verified by method (A) or (B) as above, by replacing twin-transitions by the larger set of common transitions. But even when this necessary condition is not satisfied, it may happen that a solution δ_X with a common transition does not create new critical pairs or that the newly created critical pairs can be eliminated in turn by iterating the present process a finite number of times.*

The simplest prototypical case where suppressing all critical pairs is impossible that way is when G is made up of one faulty and one unobservable transitions, both from the initial state q_0 to a state q_1, followed by an observable loop in q_1, constituting thus a critical pair. The latter transition is a twin-transition, so deferring its observable event cannot eliminate the critical pair.

Note also that a loop of an observable transition that is not a twin-transition, so an infinite succession of a same observable event in each path of a critical pair, does not pose a problem only if we take care to consider timed observations, i.e., to observe also timestamps: adding a delay block to such a transition does not change the trajectory observed when considering only order but changes it when also considering timestamp. This is the case in our example (see Figure 1) when adding a delay block to $(q_5, o1, q_5)$ or to $(q_7, o1, q_7)$ or to both: the untimed normal trajectories remain identical, and the same as the untimed faulty trajectories, i.e., $o1o2o1^\omega$ and $o3o2o1^\omega$, while the timed normal trajectories become $o1o2\#o1^\omega$ and $o3o2\#o1^\omega$, different from the timed faulty trajectories $o1o2o1^\omega\#$ and $o3o2o1^\omega\#$. In our example, only the two first of the six solutions do not need the observation of the timestamps, but just of the order of the observable events, to make the system diagnosable. But for some systems, it is impossible to make them diagnosable by simply buffering some observable events to change their order without taking into account timestamps, while it is possible when observing timestamps too. The simplest prototypical case of such a system is when G is made up of one faulty transition from the initial state q_0 to a state q_1, followed by an observable loop in q_1, and of one unobservable transition from q_0 to another state q_2, followed by an observable loop in q_2, with the same observable event o that in q_1. Adding a 1-valued delay block to this loop in q_2 provides the normal observation $\#o^\omega$ distinguishable from the faulty observation $o^\omega\#$ only if the timestamp is observed.

4 Implementation and Validation

To show the feasibility of our approach, we implemented a prototype using Python together with the SMT solver Z3. All our experimental results are obtained by running our programs on a Mac OS laptop with the processor 2.7 GHz Intel Core i5, 8 Go 1600 MHz DDR3 of memory. Source code and experiments are available at https://github.com/lu-1993/Designing-DIA-via-delay-blocks.

We reported on several versions of five benchmarks from the literature, which are all non-diagnosable DESs. The first one, ex_1, concerns the example shown in Figure 1. The second, $hvac_1$, is about the HVAC system from [9]. The third, Su_1, is a modified model from [19]. The fourth, $Mehdi_1$, is an example from [20]. And the last one, $Jiang_1$, was presented in [10]. Furthermore, considering that such literature examples are normally quite small, in order to study the scalability we tested also some hand-crafted versions whose state space was generated in a partially random way while keeping the verdicts. They are noted with 2 and 3 subscripts.

Table 1 shows part of our experimental results: the 2nd column gives the number of transitions in the system, the 3rd shows the size of the formula Ψ expressed by its num-

ber of clauses, the 4th is the min-cut value, which equals the max-flow value, the 5th is the satisfiability verdict, the 6th is the execution time in seconds when an iteration process is applied to obtain a min-cut and method (A) is used, the 7th (resp., 8th) is the execution time when the value of the max-flow is pre-computed and common transitions are dealt with by the method (A) (resp., (B)). As can be seen, our approach is feasible in practice as Z3 can get the min-cut set in the time we specified (set to 1500 seconds). In addition, it shows that the use of the min-cut value, computed in advance as the value of the max-flow, improves the efficiency (7th or 8th column compared to the 6th one), which justifies our strategy of using the max-flow min-cut theorem. Moreover, using either the method (A) or the method (B) to get rid of the common transitions provides similar efficiency, with a small advantage to the method (A) in the SAT case. It is worth noting that we have carefully chosen (and modified) the example $Jiang_1$, for which our approach cannot work for the reason explained in Section 3.4.

5 Related Work

After the diagnosability definition of DESs has been introduced, a fair amount of research has dealt with how to guarantee this property under limited sensor capacities, which is an important decision-making problem for automated systems. One well-known approach, called active diagnosis, has been initially proposed by [21]. Precisely, if a given system is not diagnosable, then a partial-observation controller is synthesized in order to force the system to stay within a diagnosable subset of its behaviors. An active diagnoser is composed of the controller and the diagnoser. After that, [22] has proposed another planning-based approach by constructing a twin plant. Then [8] has proved that the active diagnosability problem is EXPTIME-complete and proposed a way to synthesize a memory-optimal active diagnoser.

Another way to handle this problem is to start with a large amount of observable events such that the initial system is diagnosable and to calculate a subset of those observable events with minimum cardinality which ensures that the diagnosability property is still satisfied if all events outside this subset are supposed to be unobservable . This problem has been proved to be NP-complete by [23]. Then followed some work on designing a mimimal sensor set for diagnosability, that contains the original sensor set when the original system is not diagnosable ([24; 25]). However, its cardinality could be quite large and thus not very useful in practice.

Different from the above methods, our approach is applied on the existing non-diagnosable system to make it diagnosable with delay blocks in a non-intrusive way, whose complexity is polynomial.

6 Conclusion

Given a non-diagnosable system, we have shown how to find a relatively small set of transitions such that adding delay blocks to them can disambiguate all critical pairs, if possible. This is encoded in SMT as a min-cut problem on a flow network derived from the critical twin plant of the system. We have shown the efficiency of our algorithm on some benchmarks and how this efficiency is significantly improved by making use of the max-flow min-cut theorem. We have characterized the systems for which this method succeeds and we gave a sufficient condition on the system to ensure that it becomes diagnosable this way. Future research will be devoted to find a necessary and sufficient condition of diagnosability and to extend this method to the case of timed automata, handling various event occurrence times and using general delay blocks, with various delays computed in order to eliminate all critical pairs and avoid as far as possible creating new ones. From a practical point of view, we will study how finding an effective way for adding delays in the underlying system, i.e., equipping sensors with adequate software (for example, in the system of Figure 6, it is possible, even probable, that all $o2$ events labeling four different transitions are actually produced by the same sensor, in what case deferring only one of those events and not the three others will require automaton state tracking during the functioning of the system or adding in the model some way of activating or not delay of an observation).

References

[1] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[2] P. Struss. Fundamentals of Model-based Diagnosis of Dynamic Systems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 480–485. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 1997.

[3] R. Debouk, R. Malik, and B. Brandin. A Modular Architecture for Diagnosis of Discrete Event Systems. In *Proceedings of the 41st IEEE Conference on Decision and Control (CDC-02)*, volume 1, pages 417–422. IEEE., 2002.

[4] Y. Pencolé and M.-O. Cordier. A Formal Framework for the Decentralised Diagnosis of Large Scale Discrete Event Systems and Its Application to Telecommunication Networks. *Artificial Intelligence*, 164:121–170, 2005.

[5] L. Console, C. Picardi, and D. Theseider Dupré. A Framework for Decentralized Qualitative Model-based Diagnosis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 286–291. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 2007.

[6] A. Grastien, J. R. Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of Discrete-event Systems Using Satisfiability Algorithms. In *Proceedings of the 22th American National Conference on Artificial Intelligence (AAAI-07)*, pages 305–310. Menlo Park, Calif.: AAAI Press., 2007.

[7] N. Bertrand, E. Fabre, S. Haar, S. Haddad, and L. Hélouët. Active diagnosis for probabilistic systems. In Anca Muscholl, editor, *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'14)*, volume 8412 of *Lecture Notes in Computer Science*, pages 29–42, Grenoble, France, april 2014. Springer.

[8] S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. In Anil Seth and Nisheeth Vishnoi, editors, *Proceedings of the 33rd Conference on Foundations of Software Technology*

and Theoretical Computer Science (FSTTCS'13), volume 24 of *Leibniz International Proceedings in Informatics*, pages 527–539, Guwahati, India, december 2013. Leibniz-Zentrum für Informatik.

[9] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete Event System. *Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[10] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A Polynomial Time Algorithm for Testing Diagnosability of Discrete Event Systems. *Transactions on Automatic Control*, 46(8):1318–1321, 2001.

[11] J. Rintanen. Diagnosers and Diagnosability of Succinct Transition Systems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 538–544. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 2007.

[12] A. Cimatti, C. Pecheur, and R. Cavada. Formal Verification of Diagnosability via Symbolic Model Checking. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 363–369. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 2003.

[13] V. Germanos, S. Haar, V. Khomenko, and S. Schwoon. Diagnosability under weak fairness. In *Proceedings of the 14th International Conference on Application of Concurrency to System Design (ACSD'14)*, Tunis, Tunisia, june 2014. IEEE Computer Society Press.

[14] Y. Pencolé. Diagnosability Analysis of Distributed Discrete Event Systems. In *Proceedings of the 16th European Conference on Articifial Intelligent (ECAI04)*, pages 43–47. Nieuwe Hemweg: IOS Press., 2004.

[15] A. Schumann and J. Huang. A Scalable Jointree Algorithm for Diagnosability. In *Proceedings of the 23rd American National Conference on Artificial Intelligence (AAAI-08)*, pages 535–540. Menlo Park, Calif.: AAAI Press., 2008.

[16] L. Ye and P. Dague. Diagnosability Analysis of Discrete Event Systems with Autonomous Components. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, pages 105–110. Nieuwe Hemweg: IOS Press., 2010.

[17] K. Chatterjee, T.A., Henzinger, and V.S.Prabhu. Finite automata with time-delay blocks. *Proceedings of the tenth ACM international conference on Embedded software(EMSOFT '12)*, pages 43–52, 2012.

[18] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[19] X. Su, M. Zanella, and A. Grastien. Diagnosability of discrete-event systems with uncertain observations. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence IJCAI'16*, pages 1265–1271, 2016.

[20] M. Chankate, A. Philippot, V. Carre-Menetrier, and P. Marangé. Checking diagnosability on centralized model of the system. In *Proceedings of the 3rd IEEE International Conference on Control, Automation and Diagnosis ICCAD'19*, pages 386–391, 2019.

[21] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Trans. Autom. Control.*, 43(7):908–929, 1998.

[22] E. Chanthery and Y. Pencolé. Monitoring and active diagnosis for discrete-event systems. In *in Proc. SafeProcess'09*, 2009.

[23] T.-S. Yoo and S. Lafortune. NP-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Trans. Autom. Control.*, 47(9):1495–1499, 2002.

[24] L. Brandán Briones, A. Lazovik, and P. Dague. Optimizing the system observability level for diagnosability. In *Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'08)*, 2008.

[25] J. C. Basilio, S. T. S. Lima, S. Lafortune, and M. V. Moreira. Computation of minimal event bases that ensure diagnosability. *Discrete Event Dynamic Systems: Theory and Applications*, 22(3):249–292, 2012.