Diagnosability Planning for Controllable Discrete Event Systems

Hassan Ibrahim[1] and **Philippe Dague**[1] and **Alban Grastien**[2] and **Lina Ye**[1] and **Laurent Simon** [3]
[1] LRI, Univ. Paris-Sud, CNRS, Univ. Paris-Saclay, Orsay, France
Email: firstname.name@lri.fr
[2] Australian National University, Canberra, Australia
Email: alban.grastien@nicta.com.au
[3] LaBRI, Univ. Bordeaux, CNRS, Bordeaux, France
lsimon@labri.fr

Abstract

We propose in this paper an approach that aims to ensure the diagnosability of a controllable system. Given a model of correct and faulty behaviors of a partially observable discrete event system equipped with a set of elementary actions, we search a diagnosability plan, i.e. a sequence of applicable actions that leads the system from an initial belief state (a set of potentially current states) to a *diagnosable* belief state, in which the system is then left to run freely. This approach has two successive stages that keeps the diagnosability planning, including diagnosability tests, in PSPACE in comparison to the EXPTIME test for the more complex active diagnosability used usually in such cases. For this we propose to construct incrementally the twin plant structure of the given system and to exploit the parts already constructed while testing the candidate plans and constructing next parts of the general twin plant. This helps in pruning the twin plants constructions and many non diagnosability plans tests. We have created a special benchmark and tested three proposed methods with a couple of cost functions used when searching for an optimal plan.

1 Introduction

In the recent years, discrete event systems (DES) have been widely used to model and reason about large and complex systems because of their simplicity and ability to represent at a certain abstraction level real world problems in various domains [1; 2]. These systems are usually only partially observable and often subject to faults. In this context diagnosability (the property that the faults can always be detected and identified) is an important property to guarantee.

We are interested in the problem of ensuring diagnosability of a partially observable DES. Our setting is different from the classical literature (cf. Section on Related Works). We assume that the system is partially controllable through "actions" that can change its state. But we want to minimize the interactions between the controller and the system (e.g., in an electrical power system, to reduce the outage time and the wear of expensive equipment); to this end we assume that only one sequence of actions (plan) is allowed to be performed on the system. Our problem therefore translates as follows: find a plan that leads the system to a belief state where diagnosability holds.

We provide a formal definition of the problem of Diagnosability Planning. We analyze this problem and show that it is PSPACE-COMPLETE, down from EXPTIME-COMPLETENESS of active diagnosability (where control is authorized during execution). We propose a method to solve Diagnosability Planning, based on the standard twin plant technique [3; 4], to verify diagnosability. While searching for a plan, we build the twin plant to verify that the current belief state is diagnosable; we show how the twin plant constructed for a belief state can be reused for the diagnosability test of the next belief state.

The paper is organized as follows. Section 2 introduces the concepts needed: our model of controllable DES, the definition of diagnosability and its analysis by using the twin plant structure, the planning problem; and then defines the problem of diagnosability planning we address. Section 3 demonstrates complexity result for this problem. Section 4 proposes an algorithm for computing a diagnosability plan, where information about the parts of the twin plant already constructed is exploited for pruning the search and the diagnosability checking, and illustrates it on an example. Section 5 presents the construction of a scalable benchmark and experimental results of the algorithm on several of its instances. Section 6 presents related works. Section 7 concludes and draws perspectives of future research.

2 Diagnosability Planning in DES

2.1 Discrete Event Systems

This work is done in the context of DES. We assume that the system runs in two modes: the active one when the system runs freely, i.e. states are changed autonomously through partially observable transitions without any controlled exogenous event, and the reactive one in which only *feasible* exogenous actions are applied to change the system state through reactive transitions.

Definition 1. A **Controllable Labeled Transition System** (CLTS) is a tuple $G = \langle Q, \Sigma, \delta, I \rangle$ with $\Sigma = \mathcal{A} \cup \mathcal{E}$ where:

- Q is a finite set of states,

- $\mathcal{E}$ is a finite set of events,
- $\mathcal{A}$ is a finite set of actions,
- $\delta \subseteq Q \times \Sigma \times Q$ is the (active for labels in $\mathcal{E}$, reactive for labels in $\mathcal{A}$) transition relation,
- $I \subseteq Q$ is the initial belief state.

$\mathcal{E}$ is partitioned into $\{\Sigma_o, \Sigma_u, \Sigma_f\}$, where Σ_o is a finite set of observable correct events, Σ_u is a finite set of unobservable correct events, Σ_f is a finite set of unobservable faulty events.

We assume that the system is complete in terms of actions, i.e. every action is applicable in every state. This assumption can be easily made by introducing that every action inapplicable in a state q lets q unchanged and is modeled as a loop transition in q.

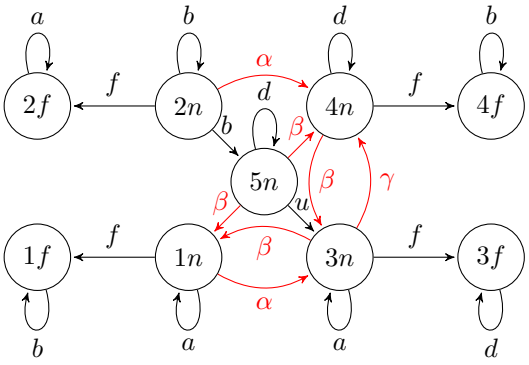

Figure 1: Illustrative Example of a CLTS

Figure 1 shows a CLTS that comprises 9 states with $I = \{1n, 2n, 3n, 4n, 5n\}$, $\mathcal{A} = \{\alpha, \beta, \gamma\}$, $\Sigma_f = \{f\}$, $\Sigma_o = \{a, b, d\}$ and $\Sigma_u = \{u\}$. In this diagram, any action that is missing in a state is assumed to leave the state unchanged (for instance, applying β in $1n$ leads to state $1n$: $(1n, \beta, 1n) \in \delta$). Notice that actions, here, do not affect the possibility of the fault occurrence.

2.2 Diagnosability in DES

Definition 2. An **active path** (or simply a path) is a sequence of active transitions, denoted by $\rho = q_0 \xrightarrow{e_1} \cdots \xrightarrow{e_n} q_n$, such that $e_1, \ldots, e_n \in \mathcal{E}$, $q_0, \ldots, q_n \in Q$ and $\forall i \in \{0, \ldots, n-1\}, (q_i, e_{i+1}, q_{i+1}) \in \delta$. The word $e_1 \ldots e_n \in \mathcal{E}^*$ is called the trajectory of ρ.

$L_I(G)$ will denote the prefix-closed language of G from the initial belief state I, i.e. the set of words from $\mathcal{E}^*$ that are the trajectories of some active paths in G that start from a state in I. Elements of $L_I(G)$ are called I-trajectories.

Definition 3. A fault $f \in \Sigma_f$ is **diagnosable** in a system G with initial belief state I if

$$\exists k \in \mathbb{N}, \forall s^f \in L_I(G), \forall t \in L_I(G)/s^f, |t| \geq k \Rightarrow$$
$$\forall \rho \in L_I(G), (P(\rho) = P(s^f.t) \Rightarrow f \in \rho).$$

In this formula, s^f denotes an I-trajectory ending by the fault f, $L_I(G)/s$ denotes the set of all extensions of s as I-trajectories and P is the projection of $\mathcal{E}^*$ on Σ_o^*, i.e. on observable events.

The above definition states that for each I-trajectory s^f ending with fault f, for each t that is an extension of s^f in G with enough (depending only on f) events, every I-trajectory ρ in G that is equivalent to $s^f.t$ in terms of observation should contain in it f. As usual, it will be assumed that $L_I(G)$ is live (i.e., for any state, there is at least one active transition issued from this state) and convergent (i.e., there is no active cycle made up only of unobservable events).

A system G is said to be diagnosable iff any fault $f \in \Sigma_f$ is diagnosable in G. We will consider only one fault type at a time, for keeping a linear complexity in the number of fault types, instead of an exponential one if they are all considered simultaneously. It will thus be assumed in the following that there exists only one fault event f ($\Sigma_f = \{f\}$), without restriction on the number of its occurrences. In the general case of n fault types, we can run our algorithm for each tested plan n times on every type of fault, and the plan is accepted iff it achieves the goal in all the n runs.

In our example, one can notice that the initial belief state is not diagnosable as it contains, for example, from the initial states $1n$ and $2n$, an arbitrary long observable sequence of the event a that might represent faulty or normal I-trajectory in the system.

Diagnosability checking of LTS is known to be polynomial in the number $|Q|$ of states [3] (actually both diagnosability definition and this complexity result are generally stated for a unique initial state q^0, i.e. $|I| = 1$, but extend easily to the general case with I of arbitrary size; as it will be seen below, the diagnosability analysis has in this case to consider at worst all pairs (q, q') of states of I instead of just one pair (q^0, q^0), i.e. the complexity is multiplied by $|I|^2$, bounded by $|Q|^2$, so remains polynomial in $|Q|$). The polynomial algorithm to check diagnosability is based on the twin plant method [3; 4]. The first step in this method is the construction from the original system model G of a non deterministic automaton, called pre-diagnoser, designed to preserve all observable information from G and to append to every state an estimate of failure information.

Definition 4. The pre-diagnoser of the system G is an observable LTS, denoted by $D = (Q_D, \Sigma_D, \delta_D, I_D)$, where:

- $Q_D \subseteq Q \times 2^{\Sigma_f}$ is the set of states (q, ℓ) with $q \in Q$ and $l \subseteq \Sigma_f$ that are reachable by δ_D from a state in I_D (see below);
- $\Sigma_D = \Sigma_o$ is the set of events;
- $\delta_D \subseteq Q_D \times \Sigma_D \times Q_D$ is the set of transitions defined by: $\delta_D = \{((q, \ell), e, (q', \ell'))$ $\in Q_D \times \Sigma_D \times Q_D | \exists$ a path $\rho = (q \xrightarrow{u_1} q_1 \ldots \xrightarrow{u_m} q_m \xrightarrow{e} q')$ in G, with $u_k \in \Sigma_u \cup \Sigma_f, \forall k \in \{1, ..., m\}$, $e \in \Sigma_o$ and $\ell' = \ell \cup (\{u_1, \ldots, u_m\} \cap \Sigma_f)$;
- $I_D = \{(q^0, \emptyset) | q^0 \in I\}$ is the set of initial states.

In the pre-diagnoser, we only keep the observable events and attach the fault information for each retained state. Precisely, if the fault has occurred up to a given state from an initial state, then the fault label for this state is f. Otherwise, it is empty. For the

illustrative example depicted in Figure 1, in its pre-diagnoser, all states ($1n$, $2n$, $3n$, $4n$, $5n$) in the initial belief state have empty as their fault label since there is no fault occurrence up to them. While the fault label for the other states ($1f$, $2f$, $3f$, $4f$) is f.

Based on the pre-diagnoser for a given system, the twin plant is then obtained by synchronizing the pre-diagnoser with itself based on the observable events, i.e., each observable event should be synchronized, to obtain all pairs of trajectories issued from initial states with the same observations.

Definition 5. The **twin plant** of the system G is the observable LTS $TP = D \parallel_{\Sigma_o} D$, where D is the pre-diagnoser of G.

Each state of the twin plant is a pair of pre-diagnoser states that provide two possible diagnoses with the same observations. Given a twin plant state, if the fault f is contained in exactly one of the two associated pre-diagnoser states, which means that the occurrence of f is not certain up to this twin plant state with the same observations, it is called an ambiguous state with respect to f. An ambiguous state cycle is a cycle containing only ambiguous states. A *critical path* is a path in the twin plant issued from a pair of initial states and made up of a prefix followed by an ambiguous state cycle. Diagnosability is verified by using the following lemma.

Lemma 1 ([3; 4])**.** *A system is non-diagnosable iff its twin plant contains a critical path.*

Now consider our illustrative example. In its twin plant, we have the path $((1n, \emptyset)(2n, \emptyset)) \xrightarrow{a} ((1n, \emptyset)(2f, f)) \xrightarrow{a} ((1n, \emptyset)(2f, f))$. This path is actually a critical path since it issued from a pair of initial states $1n$, $2n$ and contains a self cycle with an ambiguous state $((1n, \emptyset)(2f, f))$ associated with an observable event a. Thus, the system corresponding to this example is not diagnosable since one can never be sure about the fault occurrence with an arbitrary number of a observed.

2.3 Planning

Definition 6. A **plan** π for a CLTS G is a sequence of actions $a_i \in \mathcal{A}$.

Applying the plan $\pi = a_1 \dots a_m$ from a current belief state (set of states) $S \subseteq Q$ leads to the belief state $\pi(S)$ defined recursively by $\pi(S) = S$ if π is the empty sequence and $\pi(S) = \pi'(S')$ if $m > 0$, where $S' = \{q' | \exists q \in S \ (q, a_1, q') \in \delta\}$ and $\pi' = a_2 \dots a_m$. Suppose now $\pi = \alpha\beta$, for our example, $\pi(I) = \pi'(I')$, where $\pi' = \beta$ and $I' = \{3n, 4n, 5n\}$, thus $\pi(I) = \{1n, 3n, 4n\}$.

In planning, the objective is generally to find a plan that leads the system into a belief state that is included in one among a given set of "acceptable" states (e.g., states where the system is safe or where the system provides the service that we expect from it): $\pi(I) \subseteq B$ for some $B \subseteq Q$, B acceptable. The definition of the objective is not at the level of an individual state but at the level of a belief state (set of states).

Definition 7. A **planning problem** is a pair $\langle G, O \rangle$ where $G = \langle Q, \Sigma, \delta, I \rangle$ is a CLTS and $O \subseteq 2^Q$ is a collection of belief states. The solution to the planning problem is a plan π such that $\pi(I) \subseteq B$ with $B \in O$.

In our problem (formally defined next subsection), the objective is to reach a belief state where diagnosability holds.

2.4 Problem Definition

Our goal is to find a plan that leads the system to a diagnosable belief state.

Definition 8. Given a CLTS $G = \langle Q, \Sigma, \delta, I \rangle$, **diagnosability planning** is the problem of finding a plan π such that $\langle Q, \Sigma, \delta, \pi(I) \rangle$ is diagnosable.

Diagnosability planning can be equivalently phrased as the problem of solving the planning problem $\langle G, O \rangle$ where O is the set of maximal belief states from where G is diagnosable: $O = \{I' \subseteq Q | \langle Q, \Sigma, \delta, I' \rangle$ is diagnosable and I' maximal$\}$.

Our goal is to find an optimal plan for given criterion (or set of criteria). In a first phase, the criterion will be the length of the plan (number of actions in the sequence) that we want to minimize. We denote $\pi^\star$ an optimal plan. In our example, the plan $\pi^\star = \alpha\beta\beta$ is an optimal diagnosability plan that leads the system to the belief state $\pi^\star(I) = \{1n, 3n\}$ from where no critical path can be constructed.

3 Complexity

In this section, we compute the complexity of the decision problem associated with diagnosability planning. We demonstrate the following result:

Theorem 3.1. *Diagnosability planning is* PSPACE-COMPLETE.

Hardness We show it by reducing classical propositional planning to Conformant Planning (i.e. with uncertain domain, i.e. belief-space planning, as defined in Section 2.3 with $O \subseteq 2^Q$) with eXplicit States (CPXS), and then CPXS to diagnosability planning. Classical propositional planning is known to be PSPACE-COMPLETE [5]. As far as we know, the first reduction does not exist in the literature and we provide a proof sketch below.

A classical propositional planning is defined in a succinct way by a set of propositional variables and a set of actions. Each action has a precondition (subset of variables that must be true for the action to be applicable) and two sets of positive/negative effects (variables that become true/false upon the application of the action). Furthermore the problem specifies the list of variables true in the initial state and the list of variables true in the goal states. The goal is to find a sequence that leads from the initial state to one goal state.

For each variable v we create two states v_1 and v_2 that represent the fact that the variable is true or false. A belief state in the CPXS problem therefore represents a state in the classical planning problem. The initial belief state and the final objective are set accordingly For any action a and any state v_i we create a transition to v_j that models the effect of a from the perspective of variable v (to the same state if action a is not applicable). It is easy to see that the solutions for the reduced CPXS are the same as the solutions for the original classical planning problem. The second reduction can be done by making sure that no fault can occur from the CPXS objective states while the other

states have unobservable faulty cycles. The system is thus diagnosable iff the CPXS objective is reached.

Membership Membership to PSPACE can be shown by a proof similar to membership of classical propositional planning. Essentially we iterate over all the possible belief states I' (polynomial), verify that this belief state is diagnosable (can be done in polynomial time), and search for a conformant plan to this belief state in polynomial space.

4 Solving the Problem

4.1 Analyzing the problem

Solving this problem requires finding a plan that leads the system from a given belief state into a diagnosable planned belief state, called a goal state. This consists in alternating the candidate plans generation and the diagnosability test of the planned belief state of each candidate plan. Thus, one must ensure the absence of any critical path that could be constructed from this final belief state. The traditional way to do diagnosability test is just applying from scratch the twin plant approach: we call this approach *No Strategy method*. Regarding the planning steps, we have to generate the candidate plan by browsing the search space of these candidates, which could be done by any browsing algorithm, like Breadth First Search for example. The search space size can vary depending on the initial belief state and the type of available actions. For example, theoretically, if we have only one state in the initial "belief" state and we intend to build a plan which consists of deterministic actions, then the worst case of the plan size, if it exists, is $|Q|$. While if the belief state contains more than one state or the actions are not deterministic, the plan size is bounded only by $2^{|Q|}$, therefore in the worst case the plan can be encoded with $|Q|$ bits. We consider in this paper a belief state with several states and non deterministic actions.

From another hand, we notice that during the search of the intended plan, the twin plants will be constructed starting from different belief states but will generally share some states with each other. This makes interesting the idea of recycling previously built part of the general twin plant of the system. In particular, if we find any critical path during the test of a candidate plan, we know that each time we will meet again its starting state, which is the root of this constructed twin plant, we will be sure to recover a critical path. This state represents a pair of states in the source belief state. Hence the idea to label such pairs after each test in order to avoid re-testing them later if they occur in another planned belief state. Moreover, these pairs can be used to prune the next twin plants construction and even to guide the search of a diagnosability plan.

Definition 9. We call $\{q_1, q_2\}$ a **bad pair** iff the system $G = \langle Q, \Sigma, \delta, \{q_1, q_2\}\rangle$ is not diagnosable, i.e. $\{q_1, q_2\}$ is a non ambiguous starting state of a critical path in the twin plant of G. If $q_1 = q_2$, then it is called a bad unit. We denote the set of all bad pairs as $\mathcal{B}$.

In a similar way if the diagnosability test failed to find any critical path, we can call any pair represented by a state in this twin plant as a *good pair*.

Definition 10. We call $\{q_1, q_2\}$ a **good pair** iff the system $G = \langle Q, \Sigma, \delta, \{q_1, q_2\}\rangle$ is diagnosable, i.e. there is no critical path in the twin plant of G . We denote the set of all good pairs as $\mathcal{G}$.

Notice that, in our example, the pair $\{1n, 2n\}$ is a bad pair since from it a critical path can be constructed, as shown before. While $\{1n, 3n\}$ is a good pair because there is no critical path issued from it.
Consequently, one way to build the twin plant for any belief state is to process this belief state globally by adding one virtual initial state related by unobservable transitions to any state in it and to take known labeled pairs into account while constructing the traditional twin plant. We call this approach *Lazy Learning method*, as it will not exploit the good pairs: actually it will not discover them until the diagnosability plan is found where it will be too late to recycle them. However, this approach may lead to small size constructed twin plants by concentrating only on bad pairs. Another way to a better exploitation of the labeled pairs is to construct the twin plant of each pair of states in the belief state. Thus we stop the construction immediately if we meet any known bad pair and we prune developing any branch of the twin plant once we meet known good pair. We call this approach, allowing a useful recycling of good pairs, *Eager Learning method*.

4.2 Learning and exploiting Bad and Good pairs

In Diagnosability Test:
As we said above, meeting a known bad pair during a twin plant construction predicts the existence of a critical path. This allows us to completely stop the construction of the current twin plant and learn at least one new bad pair. Many other bad pairs can actually be learned, in particular all non ambiguous states in the critical path. Moreover, we can avoid testing any candidate plan that leads to a belief state that contains a known bad pair.

The *Lazy Learning method* concentrates only on exploiting the bad pairs, however the *Eager Learning method* exploits the good pairs generated after each Twin Plant construction to guide the plan generation and also to prune the further parts of Twin Plants construction. Thus, discovering a good pair means that the Twin Plant constructed starting from this pair, and using all possible output edges from each state in this pair, does not contain any critical path. This can be exploited in two ways. The first is that each state in the constructed Twin Plant represents a good pair, that can be learned. The second is that in any next Twin Plant, crossing a good pair allows pruning the construction of branches from this state (of course we have to continue the construction in other branches).

In Planning:
We can use all discovered pairs in guiding an A* algorithm for the plan generation. Let call $g(\pi)$ the cost of plan π, which is the sum of its actions costs (e.g. its length). Then we order the possible plans according to the ratio of the good pairs in their reached belief states. For this, we classify the possible plans by partitioning pairs of states in each possible next belief state $\pi(I)$ into three classes $\langle \mathcal{B}, \mathcal{U}, \mathcal{G}\rangle_\pi$, which represent

respectively bad, unlabeled and good pairs. The estimated cost function is denoted by $h(\pi)$. Then, the plan that leads to the maximum $h(\pi)$ and the minimal $g(\pi)$ is chosen. Finally, we test the goodness of $\mathcal{U}_\pi$ by the same previous iterative Twin Plant construction. Such an ordering over the possible next belief states should be updated after each test by using the discovered bad or good pairs, which will be enriched during the test of the previous plan. The process is repeated until finding a diagnosability plan or proving its absence. Notice that the labeled pairs (good and bad) will be propagated into the other classes (so there is no need to re-test any pair more than once) and will be used to prune or stop the construction of the further Twin Plants.

Observing this structure after each failed plan will provide us with information about three things; the first is the existence of diagnosability plans that can be directly deduced if ($\mathcal{B} = \emptyset \ \wedge \ \mathcal{U} = \emptyset$) in any open belief state, the second is the absence of a diagnosability plan deduced if no more candidate plans exist and *all* classes are of the form ($\mathcal{B} \neq \emptyset$) and the third is that, if none of the two stopping conditions is satisfied, the next possible test is the one on the top of the priority queue.

4.3 General algorithm

We propose the algorithm 1 which contains the general procedure that learns bad and good pairs and use them to prune the search space. This pruning is twofold: pruning the search space of the candidate plans and pruning the construction of the Twin Plant. Other procedures that add strategies for special exploitation of the bad units are also possible here.

The algorithm starts first by generating a sequence of actions as a candidate plan. This generation is done by the procedure *genCandidatePlan*. Currently, this procedure prunes tests that follow from a planned belief state I' which is a superset of a previously generated (open) belief state, i.e. I' will be closed here. Therefore no candidate plan will be generated if all nodes are closed, which is the stopping criterion for this procedure. Then, the procedure *getBestCandidate* explores the possible plan space using a cost strategy like Breadth First Search for example and optionally favors other criteria while choosing the candidate plan.

At each time a candidate plan π is generated, it is applied on the current belief state I. Then if the planned belief state $\pi(I)$ obtained does not contain any known bad pair, iteration is done on the *unlabeled* possible pairs in $\pi(I)$ in order to ensure their goodness. I.e., a pair $\{q_1, q_2\}$ is chosen and the Twin Plant $TP^{\pi}_{(q_1,q_2)}$ of the subsystem $G_{\{q_1,q_2\}}$ (having q_1 and q_2 as initial states) is built. This iteration will be broken at each time the calculated $TP^{\pi}_{(q_1,q_2)}$ has a critical path, in which case $\{q_1, q_2\}$ at least will be added to the bad pairs list. In case $TP^{\pi}_{(q_1,q_2)}$ does not contain any critical path, all pairs represented by its states will be learned as good pairs. The successors of each failed plan is done by the procedure *genCandidatePlan* and the next best candidate is chosen by the procedure *getBestCandidate*. The algorithm returns a diagnosability plan if it exists.

Algorithm 1 General algorithm

```
1: procedure FindDiagnosabilityPlan(G, I, g, h)
2:     π ← ∅; Π ← ∅
3:     B ← knownBad
4:     G ← knownGood
5:     Π ← genCandidatePlan(Π, π, G, I)
6:     π ← getBestCandidate(Π, B, G, g, h)
7:     while π ≠ ∅ do
8:         while (π(I) × π(I)) ∩ B ≠ ∅ ∧ π ≠ ∅ do
9:             π ← getBestCandidate(Π, B, G, g, h)
10:        end while
11:        if π = ∅ then
12:            return ∅
13:        end if
14:        Pairs ← getAllPossiblePairs(π(I))
15:        while Pairs \ {B ∪ G} ≠ ∅ do
16:            {q1, q2} ← chooseOne(Pairs \ {B ∪ G})
17:            DG^π_(q1,q2) ← getPreDiag(G_{q1,q2})
18:            TP^π_(q1,q2) ← getTP(DG^π_(q1,q2), Σo, B, G)
19:            Pairs ← Pairs \ {{q1, q2}, {q2, q1}}
20:            newBad ← CPPairs(TP^π_(q1,q2))
21:            if newBad ≠ ∅ then
22:                B ← B ∪ newBad
23:                break
24:            end if
25:            G ← G ∪ StatesOf(TP^π_(q1,q2))
26:            if Pairs \ {B ∪ G} = ∅ then
27:                return π
28:            end if
29:        end while
30:        Π ← genCandidatePlan(Π, π, G, I)
31:        π ← getBestCandidate(Π, B, G, g, h)
32:    end while
33:    return ∅
34: end procedure
35: procedure getBestCandidate(Π, B, G, g, h)
36:    Π ← sortByGoodness(Π, B, G, g, h)
37:    ρ ← removeTop(Π)
38:    return ρ
39: end procedure
40: procedure genCandidatePlan(Π, π, G, I)
41:    Actions ← getActions(G, π(I))
42:    while Actions ≠ ∅ do
43:        a ← removeOne(Actions)
44:        ρ ← πa
45:        if ∄ρ' ∈ Π | ρ'(I) ⊆ ρ(I) then
46:            Π ← Π ∪ ρ
47:        end if
48:    end while
49:    return Π
50: end procedure
```

4.4 Illustrative Example

We search an applicable sequence of actions that leads the system depicted in figure 1 to a diagnosable belief state. When we apply the algorithm 1, we get the Figure 2 that shows the evolution of the candidate plans and their corresponding belief states before reaching the intended plan ($\pi = \alpha\beta\beta$) that leads the system to the diagnosable state $\pi(I) = \{1n, 3n\}$. In this figure, red node represents a closed node as its belief state is a superset of previously explored node. The green node

represents the diagnosable belief state found by applying the diagnosability plan found by the algorithm 1. Other explored nodes are not diagnosable, moreover the diagnosability test is not called for any node that contains at least one known bad pair.

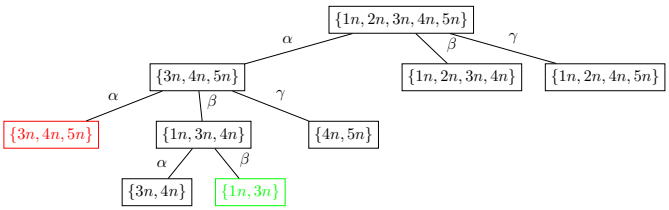

Figure 2: Plan search space, green belief state is diagnosable, others are not diagnosable, red node is closed.

5 Experimental Results

In order to test our proposed approaches on a bench mark, we have created a rectangular grid of components by repeating the active model (i.e., without its actions) of the running example in this paper and we reconfigured the actions in all components. We defined two actions models and one or the other is applied to each component according to its position in the grid. The first one is adopted for the border components and allows the planner to find a short plan, of size two or three, if the initial belief state is chosen in one of these components. The second one is adopted for the internal components and works as a relay to move from this component to the one just below or above it by using global actions that connect the action models of the components. The diagnosability plan starting from any internal component belief state is to continue moving to its below neighbors until reaching the bottom line of the grid. The tested bench does not contain any horizontal action between components in different columns. In order to connect actively by events transitions each component with its existing (at most four) neighbors, we have added an observable event c that connects each state $3n$ (resp. $1n$) at the position (i, j) in the grid to the corresponding state $2n$ (resp. $4n$) at the position $(i + 1, j)$. Similarly, we connect faulty state $3f$ (resp. $4f$) at the position (i, j) to $1f$ (resp. $2f$) at the position $(i, j + 1)$, and faulty state $1f$ (resp. $2f$) at the position (i, j) to $3f$ (resp. $4f$) at the position $(i, j - 1)$.

We have tested three search algorithms of the intended diagnosability plan. The first algorithm, the *No Strategy method*, is without any recycling and its tests are referred to by the letter N after the test numbers. The second algorithm uses what we called the *Lazy Learning method* which concentrates on learning only the bad pairs while not being able to exploit any good pair. The difference with the first algorithm is that this one learns about bad pairs and later uses them to prune another plan test or to stop another Twin Plant construction if it meets such a pair again. This approach is referred to by the letter L after the test numbers. The third algorithm is the one described in Algorithm 1. It represent what We called the *Eager Learning method*. We refer to its results in the Tables by the letter E after the test numbers.

Test Id	Matrix	$\|\mathcal{B}\|$	$\|\mathcal{G}\|$	|TPs|	|tested π|	$\|\pi\|$	Time
1N	3x3	0	0	898	38	5	0s
1L	3x3	22	3	204	18	5	0s
1E	3x3	8	6	59	17	5	0s
2N	5x3	0	0	1980	99	8	1s
2L	5x3	36	3	231	29	8	0s
2E	5x3	12	6	128	20	8	0s
3N	10x3	0	0	6127	335	14	3s
3L	10x3	113	3	444	101	14	1s
3E	10x3	33	24	612	95	14	1s
4N	20x3	0	0	51846	2350	29	27s
4L	20x3	994	3	3093	950	29	12s
4E	20x3	89	60	1227	554	29	8s
5N	50x3	0	0	1046545	38259	74	2h2m26s
5L	50x3	11336	3	108466	11252	74	17m17s
5E	50x3	219	156	9207	4320	74	8m45s

Table 1: The three methods tested on different grid heights (with width equal to 3) using BFS, where I is made up of the normal states of the central component.

Test Id	Matrix	$\|\mathcal{B}\|$	$\|\mathcal{G}\|$	|TPs|	|tested π|	$\|\pi\|$	Time
1N	3x5	0	0	7385	148	6	2s
1L	3x5	64	9	1619	56	6	1s
1E	3x5	9	12	148	24	6	0s
2N	5x5	0	0	40207	806	11	10s
2L	5x5	214	6	4429	199	11	4s
2E	5x5	25	15	529	133	11	1s
3N	10x5	0	0	325385	7118	20	1m57s
3L	10x5	2048	6	32820	2019	20	56s
3E	10x5	63	57	7518	947	20	55s
4N	15x5	0	0	3238594	43570	29	2h9m26s
4L	15x5	12087	6	4653	12051	29	20m6s
4E	15x5	105	96	56323	4260	29	10m6s
5L	17x5	34597	6	13449	34533	35	4h45m20s
5E	17x5	119	108	165193	13691	35	1h21m1s

Table 2: The three methods tested on different grid heights (with width equal to 5) using BFS, where I is made up of the normal states of the two components in the positions $(|lines|/3, 1)$ and $(2|lines|/3, 3)$.

Test Id	Matrix	$\|\mathcal{B}\|$	$\|\mathcal{G}\|$	|TPs|	|tested π|	$\|\pi\|$	Time
1L	3x3	17	3	93	14	5	0s
1E	3x3	5	3	35	6	5	0s
2L	5x3	22	3	119	16	8	0s
2E	5x3	10	6	101	12	9	0s
3L	10x3	63	3	277	50	14	1s
3E	10x3	25	21	362	39	16	1s
4L	20x3	390	3	617	353	29	5s
4E	20x3	57	48	453	149	36	3s
5L	50x3	2406	3	2255	2309	74	1m3s
5E	50x3	146	141	1119	661	96	22s

Table 3: The two learning methods tested on different grid heights (with width equal to 3) using A*, where I is made up of the normal states of the central component.

Test Id	Matrix	$\|\mathcal{B}\|$	$\|\mathcal{G}\|$	|TPs|	|tested π|	$\|\pi\|$	Time
1L	3x5	18	6	300	12	6	0s
1E	3x5	4	6	47	7	6	0s
2L	5x5	44	6	743	33	12	1s
2E	5x5	14	9	189	24	13	0s
3L	10x5	151	6	1166	133	21	3s
3E	10x5	47	54	1678	228	24	6s
4L	15x5	285	6	1634	257	30	7s
4E	15x5	52	66	1156	112	35	3s
5L	17x5	689	6	2039	653	36	18s
5E	17x5	57	75	2381	205	40	5s
6L	20x5	1005	6	2045	951	42	25s
6E	20x5	84	87	1784	215	46	5s
7L	50x5	10725	6	26226	10558	102	49m6s
7E	50x5	160	258	13413	1613	116	54s
8E	100x5	488	498	87236	15413	243	1h35m58s

Table 4: The two learning methods tested on different grid heights (with width equal to 5) using A*, where I is made up of the normal states of the two components in the positions $(|lines|/3, 1)$ and $(2|lines|/3, 3)$.

Test Id	|Comps. of belief|	$\|\mathcal{B}\|$	$\|\mathcal{G}\|$	|TPs|	|tested π|	$\|\pi\|$	Time
1N	2	0	0	153473	4904	23	1m5s
1L	2	1500	12	15916	1463	23	37s
1E	2	44	49	1384	562	23	12s
2N	3	0	0	423034	6652	27	2m14s
2L	3	1446	14	36424	1400	27	52s
2E	3	60	81	6121	750	35	36s
3N	4	0	0	594293	6254	28	2m47s
3L	4	1230	17	48385	1177	28	52s
3E	4	75	113	18524	1332	35	64s
3N	5	0	0	774272	6174	29	3m52s
3L	5	1470	20	62346	1413	29	1m1s
3E	5	74	134	15015	833	35	46s
3N	6	0	0	692229	5181	28	3m36s
3L	6	1212	23	55982	1156	28	54s
3E	6	73	139	7721	414	35	23s
4N	7	0	0	821782	5601	28	5m26s
4L	7	967	26	49440	909	28	1m
4E	7	74	144	4788	358	30	21s
5N	8	0	0	1002066	6333	28	8m1s
5L	8	1344	29	46194	1289	28	1m21s
5E	8	67	130	2795	267	29	17s
9N	9	0	0	1331933	8263	28	14m50s
9L	9	1612	32	62676	1554	28	2m6s
9E	9	56	115	3023	245	29	12s
10E	100	21	121	6340	113	35	18s

Table 5: The three methods tested on a fixed 10×10 grid size using A*, where I is made up of the normal states of an increasing number of components on the diagonal of the grid.

Successive columns in the tables show the test number along with the letter identifying the method used, the grid size (except for Table 5 where it is the number of components on which the initial belief state spreads), the number of learned bad pairs, the number of learned good pairs, the states number of the twin plant constructed, the number of tested plans, the size of the diagnosability plan computed and the CPU time.

We have first tested these algorithms on increasing size grids where the initial belief state is always made up of the five normal states in the central component of the grid. Using a Breadth First Search algorithm for the planner, Table 1 shows that the *Eager Learning method* is promising and is 15 times faster for a grid of height 50 than the classical approach that tests diagnosability without any learning. This shows the efficiency of recycling learned pairs even if they are not exploited in guiding the planning step to solve the problem.

The Table 2 is more challenging as we suppose an initial belief state composed of the normal states of two separated internal components, which are more far away when the system is bigger. The search space of the plan is much bigger, but still the performances of the two learning strategies are better than the performance of the *No Strategy method* which explodes in the test $5N$. However, results improve dramatically when exploiting the learned pairs in guiding the planning search as shown in Table 4. The benefit from this usage is also clear when comparing results in Table 3, which uses also the A* algorithm, to those in Table 1. However the heuristics used for A* here does not return an optimal length of the intended plan as does BFS strategy. But we can notice that it is very close to the optimal one for the Lazy method (which is not the case for the Eager method, although a better heuristics could be used according to the studied system structure). In order to show how changing the size of the initial belief state can affect the results, we fixed the size of the system to a grid of 10×10 and incrementally increased the size of the initial belief state by adding at each increment i the five normal states of the component at the position (i, i). Table 5 shows the interest of using learned good pairs during the search for the diagnosability plan, especially in the last lines (belief state spread on 9 components) which several hours for the two other methods.

Although the motivation of our approach is to avoid active diagnosability test in running systems, this approach is also useful at the design stage. If, after the search of a diagnosability plan, either one is found but with an expensive cost, so not favored, or no one exists, the actions should be reconfigured to achieve it. Therefore in both cases the information about the sets of good and bad pairs can be used to reconfigure the actions in order to get a plan or improve an existing one.

6 Related Works

In [6], the authors introduced the first definition of diagnosability for DESs and proposed a necessary and sufficient condition for testing it by constructing a deterministic diagnoser. The main drawback is its exponential space complexity in the number of system states. And then the authors of [3; 4] proposed new algorithms with polynomial complexity in the number of system states, which introduced the classical twin plant method. To consider real distributed systems, distributed approaches to solve diagnosability problem based on twin plant have been investigated [7; 8; 9]. However, all above approaches return only the information about whether the systems are diagnosable or not, and can do nothing for non diagnosable systems. Note that diagnosability is a quite strong property, which is often not satisfied in real systems.

On the other hand, planning techniques have been developed over the last several decades, whose idea is to find a plan that satisfies the desired goals (e.g. some given property). In [10], a synthesis method was presented that automatically generates controllers on timed transition graphs, where the specification of control requirements is expressed by metric temporal logic (MTL) formulas. In their algorithm, during searching the space of all possible paths, MTL formulas are verified over these paths to determine points at which controllable actions should be disabled to be able to get a conditional plan. It is worth noting that the authors assumed the full observability, i.e., every state variable can be observed at every step.

Considering that planning domains are often partially observable and non-deterministic, new approaches for planning under partial observability, dealing with uncertainty about the state in which the actions will be executed, were proposed in [11; 12]. The search space is no longer the set of states of the domain, but its power-set. The problem is addressed by using BDDs, which can be used to represent and efficiently manipulate the power-set of states. However, BDDs are limited to propositional representations.

A knowledge-based approach to planning is proposed in [13] with incomplete information and sensing, where plans can be built based on the available knowledge and how the knowledge state is changed by actions. At the knowledge level, one is not limited to the propositional case, i.e., the possibility exists to deal with functions

and run time variables. It should be pointed out that the various ways the physical world can be configured and how actions may change it are not taken into account. Thus, such representation cannot handle problems that require complex case analysis to distinguish various possible configurations of the world.

A close work was presented in [14], where a goal state represented in LTL is calculated and verified on the fly. In their proposed planning model, all transitions are deterministic. However, in our case, both action and event transitions are non-deterministic. Furthermore, their online learning component has, for each time, to construct a Boolean formula that can explain the violation of the considered property, which could be quite complex since different rules have to be applied at the atomic level. However, in our case, it suffices to check whether the current explored path is a critical path.

The work in [15] addressed the self healing as a combination between conformant planning and diagnosis steps to repair the system state without explicitly computing the system belief state. Given the goal states, an optimal plan is computed for a sample of the belief state that leads to a goal state, then the plan is refined depending on the result of a special diagnoser that tries to find a behavior of the system which contradicts the current plan. Once a behavior is found, this can enrich the sample to recompute a better plan. Our work can be seen as a continuation of this work where the goal states are described by the diagnosability property.

7 Conclusion and future work

In this paper, we formally define the problem of Diagnosability Planning before demonstrating that it is PSPACE-COMPLETE. Then a formal algorithm is provided to search for an optimal diagnosability plan. This is done by incrementally constructing the twin plant, during which the set of learned bad and good pairs is updated that helps in pruning the twin plant construction and in defining a cost function to guide the plan search. Experimental results demonstrate the efficiency of this approach by exploiting the different learning strategies. Considering more informative cost functions and encoding diagnosability planning in SAT is our current work. Similar approaches are also possible for planning other properties that use the twin plant structure in their verification, like predictability for example.

References

[1] C. G. Cassandras and S. Lafortune. *Introduction To Discrete Event Systems, Second Edition.* Springer, 2008.

[2] J. Vento, L. Travé-Massuyès, V. Puig, and R. Sarrate. An incremental hybrid system diagnoser automaton enhanced by discernibility properties. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 45(5):788–804, 2015.

[3] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

[4] T.-S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 47(9):1491–1495, 2002.

[5] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.

[6] M. Sampath, R. Sengupta, S. Lafortune, and K. Sinnamohideen. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 40(9):1555–1575, 1995.

[7] 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.

[8] 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.

[9] A. Schumann and Y. Pencolé. Scalable Diagnosability Checking of Event-driven System. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI07)*, pages 575–580. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 2007.

[10] M. Barveau, F. Kabanza, and R. St-Denis. A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. *IEEE Transactions on Automatic Control (TAC)*, 43(11):1543–1559, 1998.

[11] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 473–478. Morgan Kaufmann Publishers Inc., 2001.

[12] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4-5):337–384, 2006.

[13] R. P. A. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, pages 212–222, 2002.

[14] A. Ciré and A. Botea. Learning in planning with temporally extended goals and uncontrollable events. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 578–582, 2008.

[15] Alban Grastien. Self-healing as a combination of consistency checks and conformant planning problems. In *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX'15)*, pages 105–112, 2015.