DPN-SOG: A Software Tool for Fault Diagnosis of Labeled Petri Nets Using the Semi-Symbolic Diagnoser

Abderraouf Boussif, Mohamed Ghazel, Kais Klai

DPN-SOG: A Software Tool for Fault Diagnosis of Labeled Petri Nets Using the Semi-Symbolic Diagnoser

Abderraouf Boussif[1], Mohamed Ghazel[1], and Kais Klai[2]

[1] IFSTTAR, Cosys/Estas, F-59650 Villeneuve d'Ascq, France
{abderraouf.boussif, mohamed.ghazel}@ifsttar.fr
[2] Univ. Paris 13, Sorbonne Paris Cité LIPN, CNRS UMR 7030.
kais.klai@lipn.univ-paris13.fr

Abstract

In this paper, we present DPN-SOG, a software tool written in C++ for fault diagnosis of discrete event systems modeled by bounded labeled Petri nets. DPN-SOG (for **D**iagnosability analysis of **P**etri **N**ets using **S**ymbolic **O**bservation **G**raphs) implements the semi-symbolic diagnoser approach developed in [1, 2] for fault diagnosis of bounded labeled Petri nets. The implemented approach aims to cope with some limitations of the classic diagnoser-based approaches, namely the state-space explosion problem, the intermediate models and the double-checking procedure for diagnosability analysis. The key features of DPN-SOG are: (*i*) the on-the-fly building of the diagnoser and analysis of diagnosability, (*ii*) the generation of only the necessary part of the diagnoser to perform the diagnosability analysis and online diagnosis, and (*iii*) the evaluation of the time/memory consumption for the construction of the diagnoser and the analysis of diagnosability.

Contents

1 Introduction

Fault detection and diagnosis (FDD) is a crucial and challenging task, essentially in guaranteeing the reliable, safe and correct operation of complex engineered systems. To fulfill such requirements, developing effective monitoring techniques becomes essential starting from the design phase of the system. In particular, having efficient tools for monitoring and diagnosis is of great interest since this prevents or at least minimizes the failure-related down-times, especially in safety-critical systems.

From the theoretical point of view and at a high level of abstraction, discrete event systems (DES) [3], are quite suitable for fault diagnosis for a wide range of applications because of the formal basis offered by the state/transition models and their associated algorithms [4].

The early works that addressed fault diagnosis issues mostly considered finite-state automata (FSA) (the reader can refer to the recent survey [5]). Afterwards, fault diagnosis issues have also been dealt with the Petri nets (PNs) framework (an overview on PNs fault diagnosis is given in [6]).

Several software tools have been developed by the DES research community for the fault diagnosis purposes. In the context of FSA, one may consider DESUMA [7] as the reference tool for dealing with FSA fault diagnosis. In fact, DESUMA is a software developed at Michigan University. It integrates the UMDES (with a GUI for visualization) library, which is is a library of C routines for studying FSA. DESUMA allows the user to perform a variety of manipulations on FSA, such as model editing, diagnosability analysis for permanent and intermittent faults, control under full and partial observation, and decentralized control.

The LIBFAUDES library [8] is a C++ implementation designed for the analysis and synthesis of DES. Recently, it was endowed with new modules for fault diagnosis of FSA. Then, it allows analyzing event and language-diagnosis, decentralized and modular diagnosis. Some application examples are also given for the purpose of illustration.

DESLAB [9] is a scientific computing program written in PYTHON, for the development of algorithms for analysis and synthesis of FSA. It allows for the analysis of diagnosability using the verifier approach [10]. Some further tools that deal with fault diagnosis of FSA are: DIADES, DITO, MEDITO[1] and DECADA[2].

Unlike fault diagnosis of FSA, fault diagnosis of Petri net models (PNs) unfortunately lacks of tools that put the developed theory to practice. Only few tools are developed in the field.

PN_DIAG/PN_DIAG_UNB/PN_DIAG_DISC [11, 12] are MATLAB toolboxes, developed at the University of Cagliari, for the analysis of bounded/unbounded and decentralized labeled Petri nets (LPNs), respectively. PN_DIAG/PN_DIAG_DISC are now integrated in the DISC software platform[3] [13].

OF-PENDA [14] is a software tool for LPN fault diagnosis[4]. It allows for checking diagnosability and $K/K_{min}-$diagnosability using an on-the-fly and incremental approach.

We introduce here DPN-SOG[5], a software tool written in C++ for fault diagnosis of bounded LPNs. It consists in a re-implementation of OBSGRAPH tool [15], which is used for the verification of PNs using an on-the-fly model-checker based on the symbolic observation graphs. DPN-SOG implements the semi-symbolic diagnoser based approach developed in [1, 2] and allows for (i) analyzing on-the-fly diagnosability, (ii) generating a sufficient part of a semi-symbolic

[1] http://homepages.laas.fr/ypencole/softwares.html

[2] http://www.eng.iastate.edu/rkumar

[3] The software platform is being developed within the FP7 European project DISC '*Distributed supervisory control of complex plants*' (See: http://www.disc-project.eu/).

[4] OF-PENDA is developed at Ecole Centrale de Lille and IFSTTAR

[5] DPN-SOG is available from the authors upon request

diagnoser that can be used for online diagnosis and (*iii*) evaluating the required time/memory for the simultaneous generation of the diagnoser and the analysis of diagnosability.

The reminder of this paper is organized as follows. Section 2 briefly introduces some LPN notations and diagnosability analysis. Section 3 discusses the semi-symbolic diagnoser approach implemented in the tool. In Section 4, we present the features of DPN-SOG. Then, some experiments are performed on the basis of a Petri net benchmark to show the efficiency of DPN-SOG in Section 5. Finally, some concluding remarks are given in Section 6.

2 Preliminaries

2.1 Labeled Petri Net Modeling

Before going any further, we assume that the reader is familiar with Petri nets theory [16] and fault diagnosis of DES [17].

A Petri net is a structure $N = (P, T, Pre, Post)$, where P is a finite set of places; T is a finite set of transitions; Pre and $Post$ are the pre- and post-incidence mappings, respectively. $C = Post - Pre$ is the incidence matrix. A *marking* is a vector $m \in \mathbb{N}^{|P|}$ that assigns a non-negative integer to each place. We denote by $m(p)$ the marking of a place p. A marked PN (N, m_0) is a PN N with a given initial marking m_0 (it will be called PN in the sequel).

A transition t_i is *enabled* at marking m, denoted by $m[t_i >$, if $m(p) \geq Pre(p, t_i), \forall p \in P$. A transition t_i enabled at m can fire, yielding to a marking $m' = m + C \cdot (\cdot, t_i)$, where $\vec{t_i} \in \{0, 1\}^{|T|}$ is a vector in which only the entry associated with transition t_i is equal to 1. Then, m' is said to be *reachable* from m by firing t_i, denoted by $m\ [\ t_i > m'$. Moreover, a sequence of transitions $s = t_1 t_2 \dots t_k$ brings m to marking m'', denoted by $m\ [\ s > m''$, if $(\exists m_1, m_2, \cdots, m_{k-1})(m\ [\ t_1 > m_1\ [\ t_2 > \cdots\ m_{k-1}\ [\ t_k > m'')$. m'' can be computed by $m'' = m + C \cdot \pi(s)$ and denoted by $m\ [\ s > m''$, where $\pi(s) = \sum_{i=1}^{k} \vec{t_i}$ is called the firing vector relative to s. A marking m is reachable in (N, m_0) if and only if there exists a firing sequence s such that $m_0\ [s > m$. The set of all markings reachable from m_0 defines the *reachability set* of (N, m_0) and is denoted by $R(N, m_0)$.

A PN (N, m_0) is *bounded* if the number of tokens in each place does not exceed a finite number $b \in \mathbb{N}$ for any marking reachable from m_0. Formally, $\exists\, b \in \mathcal{N}$ s.t. $\forall m \in R(N, m_0), \forall p \in P : m(p) \leq b$. A PN is *live* if, no matter what marking has been reached from m_0, it is possible to fire any transition of the net by progressing through some further firing sequence [16]. Formally, $\forall m \in R(N, m_0), \forall t \in T, \exists s \in T^* : m\ [\ s.t >$.

A labeled Petri net (LPN) is a tuple $N_L = (N, m_0, \Sigma, \varphi)$, where (N, m_0) is a marked PN, Σ is a finite set of events (i.e., labels) and $\varphi\colon T \to \Sigma$ is the transition labeling function. φ is also extended to sequences of transitions, $\varphi\colon T^* \to \Sigma^*$. The language generated by N_L is $\mathcal{L}(N_L) = \{\varphi(s) \in \Sigma^* \mid s \in T^*,\ m_0\ [\ s >\}$. For short, we write $\mathcal{L}$ instead of $\mathcal{L}(N_L)$. Also, one should notice that various transitions can share the same event label, i.e., φ is not bijective. We denote by T_σ the set of transitions sharing the same event σ, i.e., $T_\sigma = \{t \in T : \varphi(t) = \sigma\}$.

2.2 Diagnosability of LPNs

Due to the partial observability, the set of transitions is partitioned as $T = T_o \uplus T_u$, where T_o is the set of observable transitions, and T_u is the set of unobservable transitions. The set of unobservable transitions is also partitioned into two subsets $T_u = T_f \uplus T_{reg}$ where T_f is the set of fault transitions while T_{reg} includes the regular (i.e., non-faulty) unobservable transitions. As we deal with labeled Petri nets, the event set Σ can also be partitioned into two disjoint sets, $\Sigma = \Sigma_o \uplus \Sigma_u$, where Σ_o is a finite set of observable events and Σ_u is a finite set of unobservable

events. Fault events denoted by set Σ_f are unobservable, thus $\Sigma_f \subseteq \Sigma_u$. Moreover, the set of unobservable events can be partitioned into two disjoint sets, $\Sigma_u = \Sigma_f \uplus \Sigma_{reg}$, where $\Sigma_{reg} = \Sigma_u \backslash \Sigma_f$ is the set of regular unobservable events, i.e., non-faulty unobservable events. In addition, the set of fault events Σ_f can be further partitioned into various fault classes, i.e., $\Sigma_f = \biguplus_{i=1}^{m} \Sigma_{f_i}$, where $\Sigma_{f_i} (i = 1, 2, \ldots, m)$ denotes the i^{th} class of faults.

Let $P_o \colon \Sigma^* \to \Sigma_o^*$ be the projection mapping which *erases* the unobservable events in any given sequence $u \in \Sigma^*$. The inverse projection operator P_{oL}^{-1} is defined as $P_{oL}^{-1}(v) = \{u \in \mathcal{L} \mid P_o(u) = v\}$ for $v \in \Sigma_o^*$. Given a live and prefix-closed language $\mathcal{L} \subseteq \Sigma^*$ and an event-sequence $u \in \mathcal{L}$, the post-language of $\mathcal{L}$ upon u denoted by $\mathcal{L}/u$ is $\mathcal{L}/u = \{v \in \Sigma^* | \ uv \in \mathcal{L}\}$. We denote by $|u|$ the length of event sequence u, and the i^{th} event of u by u^i. Also, for $a \in \Sigma$ and $u \in \Sigma^*$, we write $a \in u$ if $\exists\, i$ s.t. $u^i = a$. By abuse of notation, we note $\Sigma_{f_i} \in u$ to indicate that $\exists f_i \in \Sigma_{f_i}$ s.t. $f_i \in u$. Without loss of generality, in the sequel we will consider one single fault class Σ_f.

Definition 1. *(Diagnosability of LPNs)*
A given LPN N_L is diagnosable w.r.t. fault class Σ_f and projection P_o if:
$(\exists\, n \in \mathbb{N})\ (\forall u \in \mathcal{L}, u^{|u|} \in \Sigma_f)\ (\forall v \in \mathcal{L}/u)$*:*

$$|P_o(v)| \geq n \Rightarrow [\forall \omega \in P_{oL}^{-1}(P_o(uv)) : \Sigma_f \in \omega] \qquad \diamond$$

The above definition means that an LPN is diagnosable if for every trace u ending with a fault event (which corresponds to a fault transition) of type Σ_f, and for any sufficiently long continuation v of u, all traces ω having the same observable projection of uv contain at least one fault event. In other words, diagnosability of an LPN implies that each occurrence of a fault can be detected after a finite number of transition firings.

In what follows, we make the following assumptions:

- The LPN is deadlock-free and bounded;
- The LPN has no executable cycle of unobservable transitions and fault are assumed to be permanent.

3 The implemented approach in DPN-SOG

On the basis of the behavioral representation of PNs, *diagnoser-based* approaches are considered as the principal techniques which deal with both diagnosability analysis and online diagnosis. Nevertheless, these approaches mainly suffer from the following issues: (*i*) the state-space explosion problem that arises when constructing the diagnoser (i.e., exponential complexity) (*ii*) the use of intermediate models for constructing the diagnosers, which increases the time/memory consumption (e.g. the generator in [17], MBRG in [18] and FM-graph in [19], etc.) and (*iii*) the double-checking procedure that consists in one verification step upon the diagnoser (i.e., the existence of *F-uncertain* cycles) and the other step is performed upon the the generator or the system model (i.e., checking whether the *F-uncertain* cycle is an *F-indeterminate* one or not). Such a procedure highly increases the verification time.

The implemented approach in DPN-SOG aims to cope with the above-mentioned limitations of the diagnoser-based approaches. In fact, the approach allows for analyzing diagnosability of bounded LPNs and performing online diagnosis on the basis of a deterministic graph called SSD, for '*Semi-Symbolic Diagnoser*', derived directly from the PN model. The SSD has a particular structure, which consists in separating normal markings from faulty ones in each diagnoser node and encoding the two obtained subsets as BDDs (Binary Decision Diagrams). This aims,

in one hand, at reducing the memory requirements for saving the diagnoser and, on the other hand, at speeding up the verification process. On the basis of this structure, the approach uses a systematic procedure to check the necessary and sufficient condition for diagnosability using only the SSD with no need for any intermediate model, as it is the case of most existing diagnoser-based approaches, e.g., [17, 18, 19]. Moreover, the approach is based on an on-the-fly algorithm for simultaneously constructing the SSD and analyzing diagnosability. Thus, for non-diagnosable models, the verification process is interrupted as soon as the necessary and sufficient condition for diagnosability is violated, without building and analyzing the whole state-space of the SSD. In addition, in the case of diagnosable models, the approach generates only a part of diagnoser, which is sufficient for performing online diagnosis.

The implemented approach is inspired from the so-called *symbolic observation graph*, shortly SOG [20, 15]. The SOG was introduced as an abstraction of the reachability graph of concurrent systems and it was proved that the verification of an event-based formula of LTL/X on the SOG is equivalent to its verification on the reachability graph [15]. The main idea is to combine on-the-fly the construction and the compact representation (using BDD techniques) of the graph, in order to check LTL/X properties over finite systems. In what follows, we briefly present the SSD approach implemented in DPN-SOG.

3.1 The Semi-Symbolic Diagnoser

We introduce the following notations to define the SSD:

- Given a subset of transitions $T' \subseteq T$, we define $Enable_{T'}(m) = \{t \in T' \mid m\,[\,t >\}$, as the set of transitions in T' that are enabled at marking m. The extension to a subset of markings $M' \subseteq R(N, m_0)$, is $Enable_{T'}(M') = \bigcup_{m \in M'} Enable_{T'}(m)$.

- Given a subset of markings $M \subseteq R(N, m_0)$ and a transition $t \in T$, we define $Img(M, t) = \{m' \in R(N, m_0) \mid \exists m \in M \ : \ m\,[\,t > \ m'\}$ as the set of markings reachable from the markings in M by firing transition t. The generalization to a subset of transitions $T' \subseteq T$ is $Img(M', T') = \bigcup_{t \in T'} Img(M', t)$.

- Given a marking $m \in R(N, m_0)$ and a subset of transition $T' \subseteq T$, we define $Reach_{T'}(m) = \{m\} \cup \{m' \in R(N, m_0) | (\exists s \in T'^*) : m\,[\,s \ > \ m'\}$ as the set of markings reached by firing a sequence of transitions in T' from marking m (will be used particularly for the unobservable reachability). The generalization of this notion to a subset of markings $M \subseteq R(N, m_0)$ is $Reach_{T'}(M) = \bigcup_{m \in M} Reach_{T'}(m)$.

3.1.1 The Structure of the diagnoser node

In order to capture the main feature for analyzing diagnosability, which is to keep tracking the ambiguous behavior of the system, i.e., normal and faulty executions that share the same observable event sequence, each node in the SSD is partitioned into two subsets of markings, each of them is encoded using a BDD.

1. *the set of normal markings* (denoted by $\mathcal{M}_N$), which is the subset of markings in the node that are reachable by firing fault-free sequences.

2. *the set of faulty markings* (denoted by $\mathcal{M}_F$), which is the subset of markings in the node that are reachable by firing faulty sequences.

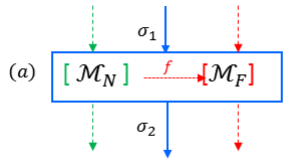

Figure 1: The general structure of the SSD node

Moreover, there may exist some faulty transitions that link some markings in $\mathcal{M}_N$ to some others in $\mathcal{M}_F$ within the same node. The existence of such transitions is also encoded within each node using a Boolean variable. The general structure of the diagnoser node is depicted in Figure 1. Actually, such a node structure can be advantageously explored for rendering diagnosability analysis more efficient than using the classical structure of diagnosers [17, 21].

One can differentiate between three types of nodes:

- *N-certain diagnoser node*: is a diagnoser node with an empty set of faulty markings ($\mathcal{M}_F = \emptyset$);
- *F-certain diagnoser node*: is a diagnoser node with an empty set of normal markings is empty ($\mathcal{M}_N = \emptyset$);
- *F-uncertain diagnoser node*: is a diagnoser node for which neither the normal set, nor the faulty set of markings, is empty, i.e., $\mathcal{M}_N \neq \emptyset$ and $\mathcal{M}_F \neq \emptyset$.

To simplify the notation, we use $a.\mathcal{M}_N$ (resp. $a.\mathcal{M}_F$) to indicate the set of normal markings $\mathcal{M}_N$ (resp. set of faulty markings $\mathcal{M}_F$) of a given diagnoser node a.

The SSD can be defined as a directed deterministic graph, where each node is composed of two BDDs encoding respectively, its subset of normal and faulty markings, while the arcs are labeled by observable events only.

Definition 2. *(Semi-Symbolic Diagnoser)*
The SSD associated with an LPN N_L is a directed deterministic graph $\mathcal{D} = \langle \Gamma, \Sigma_o, \delta_{\mathcal{D}}, \Gamma_0 \rangle$, where:

1. *Γ is a finite set of diagnoser nodes;*
2. *Σ_o is a finite set of events associated with LPN N_L;*
3. *Γ_0 is the initial diagnoser node with:*
   a) $\Gamma_0.\mathcal{M}_N = Reach_{T_{reg}}(m_0)$;
   b) $\Gamma_0.\mathcal{M}_F = Reach_{T_u}(Img(\Gamma_0.\mathcal{M}_N, T_f))$.
4. *$\delta_{\mathcal{D}} : \Gamma \times \Sigma_o \to \Gamma$ is the transition relation, defined as follows:*
   $\forall a, a' \in \Gamma,\ \sigma \in \Sigma_o$: $a' = \delta_{\mathcal{D}}(a, \sigma) \Leftrightarrow$
   $a'.\mathcal{M}_N = Reach_{T_{reg}}(Img(a.\mathcal{M}_N, T_\sigma)) \wedge$
   $a'.\mathcal{M}_F = Reach_{T_u}(Img(a'.\mathcal{M}_N, T_f) \cup Img(a.\mathcal{M}_F, T_\sigma))$. ◇

3.2 Diagnosability Analysis

The authors in [17] have established a necessary and sufficient condition for diagnosability on the basis of the generator/diagnoser models. The same condition has been reformulated in [18] for analyzing diagnosability of bounded LPNs.

In the SSD approach, a necessary and sufficient condition for diagnosability is formulated on the basis of the SSD structure and a systematic procedure for checking such a condition on the fly and directly upon the diagnoser is developed. These main features rely on some theoretical results that are briefly recalled in what follows. It should be noticed that for the sake of space, all proofs of propositions and theorems introduced in this section are omitted and can be found in [2].

Proposition 1. *Let $c\ell = a_1, a_2, \ldots, a_n$ be an F-uncertain cycle in $\mathcal{D}$, with $\delta_{\mathcal{D}}(a_i, \sigma_i) = a_{(i+1)mod_n}$ for $1 \leq i \leq n$. Then, there exists at least one fault-free cycle in LPN N_L that shares the same observation $(\sigma_1, \sigma_2, \ldots, \sigma_n)^*$.* ◇

This result is interesting for checking *F-indeterminate* cycles, using any diagnoser-based approach. It is, in fact, sufficient to check that an *F-uncertain* cycle in the diagnoser corresponds to a faulty cycle in the original model (or the intermediate model), without needing to check the existence of a faulty-free cycle.

3.2.1 Necessary and Sufficient Condition

In the SSD approach, the necessary and sufficient condition for diagnosability is established on the basis of the notion of '*indicating sequence*', which is associated with the F-uncertain cycles.

Definition 3. *($c\ell$-indicating sequence)*
Let $c\ell = a_1, a_2, \ldots, a_n$ be an F-uncertain cycle in $\mathcal{D}$ (the starting node a_1 can be arbitrarily chosen in the cycle), with $\delta_{\mathcal{D}}(a_i, \sigma_i) = a_{(i+1)mod_n}$ for $1 \leq i \leq n$. $c\ell$-indicating sequence $\rho^{c\ell} = \mathcal{S}_1, \mathcal{S}_2, \ldots,$ is an infinite sequence of sets of markings, such that:
– $\mathcal{S}_1 = a_1.\mathcal{M}_F$;
– $\forall\, i > 1 : \mathcal{S}_i = Reach_{T_u}(Img(\mathcal{S}_{i-1}, T_{\sigma_{(i-1)_{mod_n}}}))$. ◇

In fact, the $c\ell$-indicating sequence tracks the subsets of faulty markings in each node of $c\ell$ without considering the faulty markings generated through the occurrence of some faulty transitions outgoing from the normal set of markings in the traversed nodes (except for $\mathcal{S}_1$ which holds all the faulty markings of $a_1.\mathcal{M}_F$, i.e., $\mathcal{S}_1 = a_1.\mathcal{M}_F$).

Theorem 1. *For an F-uncertain cycle $c\ell = a_1, a_2, \ldots, a_n$ in $\mathcal{D}$, and $\rho^{c\ell} = \mathcal{S}_1, \mathcal{S}_2, \ldots$ its corresponding $c\ell$-indicating sequence. Then, $c\ell$ is an F-indeterminate cycle if and only if $\forall i \in \mathbb{N}^* : \mathcal{S}_i \neq \emptyset$.* ◇

Actually, Theorem 1 states that an F-uncertain cycle is an F-indeterminate one if the $c\ell$-indicating sequence does not reach an empty fixed-point (see [2]).

3.2.2 A Procedure for Checking Diagnosability

For the actual verification of diagnosability, *a systematic procedure* is derived directly from Theorem 1 and can be performed as follows:

When an F-uncertain cycle $c\ell$ is found in SSD $\mathcal{D}$, then:

1. generate the successive elements of $c\ell$-indicating sequence $\rho^{c\ell}$ (starting from $\mathcal{S}_1$), and for each element $\mathcal{S}_i$ check the following conditions:

(a) if $\mathcal{S}_i = \emptyset$, then cycle $c\ell$ is not an F-indeterminate cycle and therefore the procedure is stopped;

(b) else, if $\mathcal{S}_i \neq \emptyset$ and $\exists k \in \mathbb{N} : i = 1 + kn$ (with $n = |c\ell|$), then:

i. if $\mathcal{S}_i = \mathcal{S}_{(i-n)}$, then cycle $c\ell$ is an F-indeterminate cycle and stop the procedure;

ii. else continue.

This procedure is repeated on each F-uncertain cycle generated on the fly in $\mathcal{D}$.

4 Features of DPN-SOG tool

DPN-SOG consists in a modified and re-implemented version of ObsGraph tool [15], which is a BDD-based tool implementing various verification approaches for workflows/PNs using symbolic observation graph through an on-the-fly model-checker.

DPN-SOG is a command-line software tool developed in C++ programming language (available for Linux). It integrates the BDD package Buddy [22] which is used for symbolic manipulation of automata and other computation models. Actually, DPN-SOG takes as inputs: (*i*) the LPN models in prod format [23], (*ii*) the bound k of the net as an integer and (*iii*) a text file which specifies the sets of observable, non-observable and faulty transitions. Using these ingredients, DPN-SOG builds on the fly the SSD and simultaneously analyzes the diagnosability. When the LPN model is non-diagnosable, DPN-SOG outputs the generated part of the diagnoser as well as a witness diagnoser event-trace that violates the diagnosability property (the first encountered event-trace). When the LPN model is stated to be diagnosable, DPN-SOG generates the part of the diagnoser that is sufficient to perform the online diagnosis. Further complementary information that help the evaluation of the approach can be output, namely, the required CPU time, the size of the diagnoser, the number of used BDD nodes and memory size of the diagnoser (in kilobytes).

5 Experimental Evaluation of DPN-SOG

In order to evaluate DPN-SOG tool (and thus the implemented approach), some experimentations have been performed using a parametric PN benchmark, which illustrate the concept of permanent fault and fulfills the assumptions considered by the approach.

The PN benchmark, depicted in Figure 2, describes a manufacturing plant characterized by three parameters: m, k and b, where:

- k is the number of production lines;
- m is the number of units of the final product that can be simultaneously produced, while each unit is composed of k parts;
- b is the number of operations that each part must undergo in each line.

The faulty transitions are indicated by red boxes, while the other transitions (observable or unobservable depending on the experiments we will carry out) are in gray.

The obtained results are discussed with respect to a reference approach for fault diagnosis of LPNs, called MBRG/BRD technique [21]. In fact, the MBRG/BRD technique is implemented as a Matlab Toolbox called PN_DIAG tool [11]. It allows generating two models, namely the MBRG and the BRD graphs, which are respectively more or less equivalent to the generator

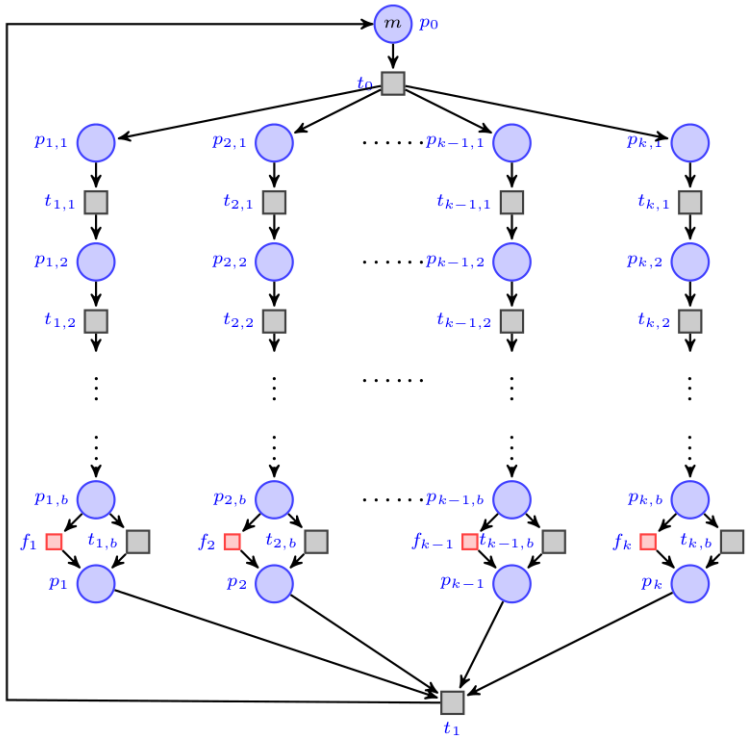

Figure 2: The PN benchamrk

and the diagnoser in [17], but for PNs. The diagnosability analysis is then performed based on these models.

In this work, two series of tests are considered:

Series (1): we evaluate the efficiency of our approach while varying the number of observable and unobservable transitions in the model. Thus, we consider the following parameters: $m = 1$, $b = 10$, and $k = 4$. Transitions t_0 and t_1 are observable while transitions $f_i \in T_f$ are faulty (for $1 \leq i \leq 4$). Regarding transitions $t_{i,10}$ (i.e., transitions in parallel with faulty transitions), two cases are considered:

Test 1: transitions $t_{i,10}$ are non-observable (for $1 \leq i \leq 4$), in this case the model is non-diagnosable;

Test 2: transitions $t_{i,10}$ are observable (for $1 \leq i \leq 4$), in this case the model is diagnosable.

Concerning the reset of transitions, we first consider them unobservable and then after each simulation, we increase the number of observable transitions in the model, i.e., we increment the number of observable transitions after each simulation, from 2, 6,..., until 38 observable transitions, for Test 1 and from 6, 10,..., until 42 observable transitions for Test 2. Finally, it should be noticed that for the sake of clarity the transition labels and names are the same (i.e., $\varphi(t_0) = t_0$).

Series (2): we evaluate the memory efficiency of our approach regarding the ratio of observable and unobservable transitions in the PN models. In this series, we consider that parameters $m = 2$, $k = 4$, and $b = 2 \ldots, 10$. Transitions t_0 and t_1 are observable. Transitions

f_i and $t_{i,b}$ for $1 \leq i \leq 4$ are unobservable. Regarding the rest of transitions two cases are considered:

a) all the other transitions are unobservable;

b) all the other transitions are observable.

In this last series, we are only interested in the construction of the diagnoser, without the verification of diagnosability.

5.1 Experimental Results

The experiments have been performed on 64-bit PC (CPU: Intel Core i5, 2.5 GHz, RAM: 6GB). We fix 4 hours as a maximum analysis duration above which we consider that the tool failed to return a result. The experimental results are summarized in Table 1 for Series (1) and Table 2 for Series (2), where:

— *Obs* is the number of observable transitions in the considered model;

— $|M_S|$ and $|M_T|$, are the number of states and transitions of the reachability graph respectively;

— $|\mathcal{D}_S|$ and $|\mathcal{D}_T|$ are, respectively, the numbers of nodes and arcs in the SSD;

— $\mathcal{D}_e$ and $\mathcal{D}_m$ are, respectively, the elapsed time (in seconds) for generating the SSD and analyzing diagnosability on the fly, and the memory required for building the SSD (in kilobytes);

— $|G1_S|$, $|G1_T|$ and $G1_e$ are, respectively, the numbers of states and transitions in the $MBRG$ and the elapsed time for constructing the $MBRG$;

— $|G2_S|$, $|G2_T|$ and $G2_e$ are, respectively, the numbers of states and transitions in the BRD and the elapsed time for constructing the BRD;

— 'T_e' is the time required for giving diagnosability verdict using MBRG/BRD approach.

5.2 Discussion

In this section, we highlight the main observations that can be derived from the obtained results. Firstly, by considering $m = 1$, $k = 4$ and $b = 10$, the LPN benchmark has 45 places, 46 transitions, while its reachability graph contains 14642 markings and 58566 transitions.

5.2.1 Time Efficiency of the SSD Approach from Series (1)

- In the case of non-diagnosable models (i.e., Test 1), one can observe that our tool efficiently analyzes the diagnosability by only constructing the relevant part of the diagnoser. Actually, as soon as an F-indeterminate cycle is found, the construction/verification process is stopped and the model is stated to be non-diagnosable. Consequently, the diagnosability verdicts are given in less than one second even for large values of *Obs*.
- In the case of diagnosable models (i.e., Test 2), the SSD approach potentially needs to generate a larger part of the diagnoser state-space. Consequently, the verification process checks all the F-uncertain cycles that exist in the diagnoser. Thanks to the systematic procedure for checking diagnosability, the verification time is not too much affected.

Table 1: Experimental Comparative Results for Series (1)

	SSD			MBRG			BRD				
Obs	$\|\mathcal{D}_S\|$	$\|\mathcal{D}_T\|$	$\mathcal{D}_e(s)$	$\|G1_S\|$	$\|G1_T\|$	$G1_e(s)$	$\|G2_S\|$	$\|G2_T\|$	$G2_e(s)$	T_e	
10	20	19	$\leq$ 0.1	257	785	2	164	436	1	a.q.	Non-diagnosable
14	28	27	$\leq$ 0.1	626	2017	6	514	1540	10	a.q.	
18	36	35	$\leq$ 0.1	1297	4337	17	1252	4004	68	a.q.	
22	44	43	$\leq$ 0.1	2402	8249	47	2594	8644	419	a.q.	
26	52	51	$\leq$ 0.1	4097	14353	120	4804	16468	1493	-	
30	60	59	$\leq$ 0.1	6562	23345	289	8194	28676	4543	-	
34	68	67	$\leq$ 0.1	10001	36017	685	*	*	o.t.	-	
14	257	770	0.6	257	1026	2.28	515	1571	8	a.q.	Diagnosable
18	626	2002	0.8	626	2502	7.15	1253	4035	53.6	a.q.	
22	1297	4322	1	1297	9606	20.29	2595	8675	261.1	a.q.	
26	2402	8234	1.1	2402	9606	55.18	4805	16499	1084.5	a.q.	
30	4097	14338	1.6	4097	16386	155	8194	28707	3489	-	
34	6562	23330	1.8	6562	26246	371	13125	46691	9991	-	
38	10001	36002	2.5	10001	40002	721	*	*	o.t.	-	

*: No result obtained o.t.: Out of time ($\geq$ 4 hours). a.q.: '*accident quit*'

Table 2: Experimental results for series (2)

b	$\|M_S\|$	$\|M_T\|$	$\|\mathcal{D}_S\|$	$\|\mathcal{D}_T\|$	$\mathcal{D}_e$(s)	$\mathcal{D}_m$(kb)
2	1378	8264	6	8	$\leq$ 0.1	27
3	10257	65538	6	8	$\leq$ 0.1	50
4	51251	341252	6	8	0.1	85
5	195778	1341362	6	8	$\leq$ 0.2	128
6	617058	4317000	6	8	0.2	152
7	1683713	11658514	6	8	$\leq$ 0.3	174
2	1378	8264	196	564	0.16	696
3	10257	65538	2756	11128	1.2	11916
4	51251	341252	20514	98564	18.9	112823
5	195778	1341362	102502	546504	330	674500
6	617058	4317000	391556	2236470	4051	2857630
7	1683713	11658514	*	*	o.t.	*

5.2.2 A Comparative Analysis with the MBRG/BRD Approach

- In the case of non-diagnosable models (i.e., Test 1), our approach is largely more efficient compared to the MBRG/BRD approach. This is basically due to the fact that our approach is based on an on-the-fly procedure for generating the diagnoser and analyzing diagnosability.
- Our approach remains more efficient when there is a large number of unobservable transitions compared to the MBRG/BRD approach in the case of diagnosable models (Test 2). This may be explained through three points:

  1. Our approach only constructs one graph since diagnosability analysis is performed

directly on the diagnoser;

2. The systematic procedure for checking the F-indeterminate cycles allows reducing the verification time compared to the MBRG/BRD;
3. Our approach generates only the necessary part of the SSD for analyzing diagnosability contrarily to the MBRG/BRD approach, where the whole state-spaces of the MBRG/BRD are generated.

- As shown in Table 1, when the model contains more than 30 observable transitions, PN_Diag tool spends more than 4 hours without generating the BRD and thus without deciding the diagnosability. However, our tool achieves the task in few seconds.
- Regarding the diagnosability analysis using PN_DIAG tool, some *accident quits* occur during its running (i.e., an exit without any output results). This may be caused by eventual bugs in the tool. Thus, we do not compare the elapsed time for checking diagnosability.

5.2.3 Memory Efficiency of the SSD Approach from Series (2)

- The needed memory for representing the SSD is measured by considering the memory needed for representing one BDD node, which is fixed by the used BuDDy library in ObsGraphTool to 20kbytes, multiplied by the total number of nodes in all the bdds that represent the diagnoser.
- the number of nodes/transitions of the SSD is not affected by increasing the number of the unobservable transitions. However, it is very sensitive to the number of observable transitions.
- the SSD spends more computing time to construct the diagnoser when the number of observable transition increase than when the number of unobservable transitions increases. This is a consistent consequence for the increasing of the state-space size in this case.
- the symbolic representation leads to an important memory saving when the model contains a large number of unobservable transitions. That is, when the SSD nodes contain a large number of markings (which corresponds to the case of a large number of unobservable transitions in the model), then the corresponding BDDs will be efficiently compacted. This is due to the fact that BDDs are particularly convenient to represent large sets of markings. However, when the SSD nodes contain a few number of markings, it is more difficult to compact them using BDDs. Thus, this explains the considerable memory consumption when the model contains a large number of observable transitions. In fact, when the number of observable transitions increases in the model, the SSD converges to the classic diagnosers [17, 18] in terms of memory required for the diagnoser construction, which decreases the efficiency of the symbolic representation.

6 Conclusion

DPN-SOG is a software tool for fault diagnosis of bounded labeled Petri nets, on the basis of the semi-symbolic diagnoser approach developed in [1, 2]. DPN-SOG serves to analyze diagnosability on the fly of LPNs, generating only the necessary part of the diagnoser for performing the diagnosability analysis and online diagnosis and evaluating the memory/time consumption. As future works, we intend to endow DPN-SOG with a graphic interface in order

to facilitate the use of the tool. Moreover, we wish to extend the tool in order to deal also with quantitative properties of diagnosability (K/K_{min}-diagnosability). Finally, we are working on a modular version of the semi-symbolic diagnoser approach, that will be integrated in the tool.

Acknowledgments

The authors acknowledge the support of the ELSAT2020 project. ELSAT2020 is co-financed by the European Union with the European Regional development Fund, the French state and the Hauts-de-France Region Council.

References

[1] Abderraouf Boussif, Mohamed Ghazel, and Kais Klai. Combining enumerative and symbolic techniques for diagnosis of discrete-event systems. *Verification and Evaluation of Computer and Communication Systems*, pages 1–15, 2015.

[2] Abderraouf Boussif. Contributions to fault diagnosis of discrete-event systems. *Ph.D. Thesis, IFSTTAR*, 2016.

[3] C.G. Cassandras and S. Lafortune. Introduction to discrete event systems. *Springer*, 2008.

[4] F. Lin. Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems*, 4(2):197–212, 1994.

[5] J. Zaytoon and S. Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

[6] Francesco Basile. Overview of fault diagnosis methods based on petri net models. *European Control Conference (ECC)*, pages 2636–2642, 2014.

[7] L Ricker, S Lafortune, and S Genc. DESUMA−a tool integrating giddes and UMDES. In 8^{th} *international workshop on discrete-event systems*, 2006.

[8] Thomas Moor, Klaus Schmidt, and Sebastian Perk. libfaudes -an open source C++ library for discrete event systems. In 9^{th} *International Workshop on Discrete Event Systems*, pages 125–130. IEEE, 2008.

[9] Leonardo B Clavijo, João Carlos Basilio, and Lilian Kawakami Carvalho. Deslab: A scientific computing program for analysis and synthesis of discrete-event systems. 11^{th} *IFAC Workshop on Discrete Event Systems*, 45(29):349–355, 2012.

[10] M.V. Moreira, T.C. Jesus, and J.C. Basilio. Polynomial time verification of decentralized diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 56(7):1679–1684, 2011.

[11] M. Pocci. A toolbox for diagnosability of petri nets. *http://www.diee.unica.it/giua/TESI/09_Marco.Pocci/*, 2009.

[12] Maria Paola Cabasino, Alessandro Giua, Laura Marcias, and Carla Seatzu. A comparison among tools for the diagnosability of discrete event systems. *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 218–223, 2012.

[13] Maria Paola Cabasino, L Contini, Alessandro Giua, Carla Seatzu, and A Solinas. A software platform for the integration of discrete event systems tools. In *Automation Science and Engineering (CASE), 2011 IEEE Conference on*, pages 45–51, 2011.

[14] Baisi Liu, Mohamed Ghazel, and Armand Toguyéni. OF-PENDA: A software tool for fault diagnosis of discrete event systems modeled by labeled petri nets. In *ADECSPetri Nets*, pages 20–35, 2014.

[15] K. Klai and D. Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. *Proceedings of the* 29^{th} *International Conference on Application and Theory of Petri Nets*, pages 23–27, 2008.

[16] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

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

[18] M. P. Cabasino, A. Giua, and C. Seatzu. Diagnosability of bounded Petri nets. *Proceedings of the* 48^{th} *IEEE Conference on Decision and Control, held jointly with the* 28^{th} *Chinese Control Conference*, pages 1254–1260, 2009.

[19] B. Liu. An efficient approach for diagnosability and diagnosis of DES based on LPN. *PhD thesis, Ecole Centrale de Lille*, 2014.

[20] S. Haddad, J.-M. Ilié, K. Klai, and F. Wang. Design and evaluation of a symbolic and abstraction-based Model Checker. 2^{nd} *International Symposium on Automated Technology for Verification and Analysis (ATVA'04)*, pages 198–210, 2004.

[21] M. P. Cabasino, A. Giua, and C. Seatzu. Diagnosability of discrete-event systems using labeled petri nets. *IEEE Transactions on Automation Science and Engineering*, 11(1):144–153, 2014.

[22] J Lind-Nielsen. BDD package buddy, v. 1.9. *http://buddy.sourceforge.net/manual/*, 2000.

[23] Kimmo Varpaaniemi, Keijo Heljanko, and Johan Lilius. Prod 3.2 an advanced tool for efficient reachability analysis. *International Conference on Computer Aided Verification*, pages 472–475, 1997.