OF-PENDA: A Software Tool for Fault Diagnosis of Discrete Event Systems Modeled by Labeled Petri Nets

Baisi Liu[1,2,3], Mohamed Ghazel[1,2], and Armand Toguyéni[1,3]

[1] Univ. Lille Nord de France, F-59000, Lille, France
[2] IFSTTAR, Cosys/Estas, F-59666, Villeneuve d'Ascq, France
[3] École Centrale de Lille, LAGIS, F-59651, Villeneuve d'Ascq, France
{baisi.liu, mohamed.ghazel}@ifsttar.fr, armand.toguyeni@ec-lille.fr

Abstract. In this paper, a software tool to deal with diagnosis of discrete event systems (DESs) is presented. This tool called On-the-Fly PEtri-Net-based Diagnosability Analyzer (OF-PENDA) implements the techniques developed in [13] for the diagnosis of DESs modeled by labeled Petri nets (LPNs). This technique aims to cope with the state explosion problem which is a major issue when dealing with diagnosis of DESs. In particular, OF-PENDA implements an incremental and on-the-fly algorithm which makes it possible to analyze (K-)diagnosability without necessarily generating the whole state space of the model. Three aspects for OF-PENDA are discussed in this paper: an overview on the implemented technique is given; some features of the tool are discussed; then two illustrative case studies are processed to show the efficiency in terms of time and memory compared with some existing approaches.

Keywords: fault diagnosis, discrete event system, labeled Petri net, on-the-fly analysis, incremental search, C++.

1 Introduction

Diagnosability analysis and online diagnosis are two crucial issues in safety critical systems. Most of such systems can be modeled as DESs [5] in a sufficiently high abstraction level. For technical and/or economical reasons, it is generally inconceivable to sense all the behavioral variations in a complex dynamic system. Thereby, very often, monitoring activities in such systems have to be carried out under partial observability. In particular, DES diagnosis is often explained as the task which consists in deducing and identifying the occurrence of faulty unobservable events, based on the occurrence of observable events; and diagnosability is the ability to diagnose any fault in a finite delay after its occurrence.

In the past two decades, diagnosability analysis has been studied using techniques such as diagnoser automata [15], verifier automata [7, 17], integer linear programming [1] and verifier Petri nets [3]. Software tools, such as the UMDES library [9] can analyze diagnosability. A major issue that these works have to

deal with is the computation complexity. In particular, the state explosion phenomenon constitutes a major obstacle especially for the approaches based on building and/or investigation of the state space. In [11], we have developed a new technique which aims to cope (even partially) with this problem, by adapting an incremental and on-the-fly building of the state space in parallel with (K-)diagnosability analysis.

OF-PENDA is a software tool developed in C++ for the diagnosability analysis and the online diagnoser generation for DESs modeled by LPNs. It is the implementation of various algorithms developed in [11]. Diagnosability verdict is emitted and when the system is diagnosable, an online diagnoser is generated in a straightforward way. In addition, for diagnosable systems, K_{min}, the minimum number of steps (observable events) after the occurrence of fault to ensure diagnosability, is also given to help describe diagnosability in a finer way.

The reminder of this paper is organized as follows. Section 2 briefly introduces some notations pertaining to LPN and (K)-diagnosability. Section 3 discusses the on-the-fly and incremental techniques implementated in OF-PENDA. In Section 4, we present the features of OF-PENDA. Then, two cases WODES and level crossing (LC) benchmarks are used to show the efficiency of OF-PENDA in Section 5 and 6, respectively. Finally, some concluding remarks are given in Section 7.

2 Preliminaries

2.1 Labeled Petri Nets

A Petri net (PN) is a tuple $N = (P, T, Pre, Post)$, 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. $C = Post - Pre$ is the incidence matrix. A marking is a vector $M \in N^{|P|}$ that assigns a non-negative integer to each place. A marked PN (N, M_0) is a PN N with given initial marking M_0. For short, a marked PN will be called PN afterward. A transition t_i is enabled at marking M if $M \geq Pre(\cdot, t_i)$, denoted by $M\ [\ t_i >$.

A PN (N, M_0) is bounded if the number of tokens in each place does not exceed a finite number $m \in N$ for any marking reachable from M_0. A PN is live if, no matter what marking has been reached from M_0, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence.

An 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 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 $\mathcal{L}(N_L) = \{\varphi(\sigma) \in \Sigma^* \mid \sigma \in T^*,\ M_0\ [\ \sigma >\}$. For short, we write $\mathcal{L}$ instead of $\mathcal{L}(N_L)$. We denote by $s = s_1 s_2 \ldots s_n$ the concatenation of $s_1, s_2, \ldots, s_n$, where $s_1, s_2, \ldots, s_n \in \Sigma^*$.

2.2 *K*-diagnosability

In event-based diagnosis of DESs, event set Σ is 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 are unobservable, thus the set of fault events $\Sigma_f \subseteq \Sigma_u$. Likewise, the set of transitions T is partitioned into sets of observable and unobservable transitions: $T = T_o \uplus T_u$. Moreover, the faulty transitions are unobservable ($T_f \subseteq T_u$). Also, the set of fault events can be partitioned into m sets, $\Sigma_f = \biguplus_{i=1}^{m} \Sigma_{F_i}$, where Σ_{F_i} denotes one class of faults.

Let $P_o\colon \Sigma^* \to \Sigma_o^*$ be the projection which "erases" the unobservable events in a sequence $s \in \Sigma^*$. The inverse projection operator P_o^{-1} is defined as $P_o^{-1}(r) = \{s \in \mathcal{L} \mid P_o(s) = r\}$ for $r \in \Sigma_o^*$. Given a live and prefix-closed language $\mathcal{L} \subseteq \Sigma^*$ and a string $s \in \mathcal{L}$, the post-language of $\mathcal{L}$ after s denoted by $\mathcal{L}/s$ is $\mathcal{L}/s = \{s' \in \Sigma^* | \ ss' \in \mathcal{L}\}$. We denote the length of string s by $|s|$, and the i^{th} event of sequence s by s^i. For $a \in \Sigma$ and $s \in \Sigma^*$, we write $a \in s$ if i exists such that $s^i = a$.

Definition 1. *Given an LPN N_L and $K \in N$, $e \in \Sigma_f$ is K-diagnosable if $\forall u \in \mathcal{L}, u^{|u|} \in \Sigma_f$ and $\forall v \in \mathcal{L}/u$ such that $|P_o(v)| \geq K$, then*

$$r \in P_o^{-1}(P_o(uv)) \Rightarrow e \in r$$

For a K-diagnosable system S, it is obvious that S is K'-diagnosable for any $K' \geq K$. Besides, K_{min} exists such that S is K_{min}-diagnosable, and S is not K'-diagnosable for all $K' < K_{min}$.

3 Techniques Applied in OF-PENDA

3.1 On-the-fly Analysis

For most existing approaches, diagnosability analysis is composed of two stages. First, advanced models are developed for extracting the necessary information for diagnosability analysis from the original model, e.g., diagnoser automata [15], verifier automata [17], verifier Petri nets [3], and linear inequalities [1]. Secondly, diagnosability analysis is tackled based on the structure analysis of the plant (or by solving mathematical models), e.g., through checking the existence of certain specific states or cycles, and verifying the existence of linear inequalities solutions. Traditionally, the two stages are proceeded independently and sequentially. The analysis of advanced models is performed after their state spaces have been completely generated. This presents the state explosion problem when dealing with large systems. In OF-PENDA, we tackle this problem by using on-the-fly techniques [16], since such techniques have the following advantages:

On-the-fly techniques can save memory resources. Generally, on-the-fly techniques permit us to generate and investigate only a part of the state space to find solutions, unlike the classic enumerative approaches which have to build the whole state space *a priori*. On-the-fly exploration techniques do not reduce

the complexity of the original algorithms, but they do save memory resources in general, depending on the system structure and on the searching strategy.

On-the-fly techniques can save computing time. On the one hand, on-the-fly exploration terminates as soon as some specific features are found, which requires less time than investigating the whole state space. On the other hand, for two-stage analysis, such as our approaches that will be given in this paper, the advanced models can be derived and analyzed step by step as the on-the-fly building of the basic models, rather than being analyzed after building the whole basic models, which can save time from both analysis stages.

On-the-fly techniques can deal with some unbounded systems, since they return a verdict as soon as some specific features are found, instead of investigating the whole infinite state space, as will be shown in Section 5.

3.2 Incremental Search Technique

The incremental method [8] is a search technique that reuses the information from previous searches when some parameters change. Generally, it is faster than performing the search for each changed parameter from scratch. The analysis of the k^{th} step is based on the search result of the $(k-1)^{th}$ step. Different from other speeding up searches, it can guarantee finding the shortest paths.

As in [13], the classic diagnosability can be analyzed based on incrementally investigating K-diagnosability with increasing the value of K, and the K_{min} value which ensures the diagnosability will be eventually found (for diagnosable systems). Note that some approaches based on integer programming [1] have to rebuild and solve the equation system (or inequalities) when seeking out K_{min}, without using the previous search results.

The incremental technique is a skillful technique to speed up the search procedure. It should be used for bounded systems so that the search can terminate well. In particular, it can be used for unbounded systems when some conditions for terminating the search exist.

Incremental techniques can be used with on-the-fly analysis to perform an efficient analysis. Both techniques do not change the computation complexity. However, they improve the searching efficiency when dealing with real systems.

4 Features of OF-PENDA

In [13], we deal with diagnosability of LPN models using on-the-fly and incremental techniques. Classic diagnosability is transformed into a series of K-diagnosability, where K increases progressively. Unlike several existing approaches [7,15,17], we do not enumerate the state space. Instead, the state space is often partially generated as necessary, and the investigation is stopped as soon as some special criteria are met. This can potentially offer efficiency in terms of time and memory, compared with the aforementioned approaches.

OF-PENDA is the implementation of various algorithms elaborated in [13] for the diagnosis of DESs modeled by LPNs. It is a command-line software tool developed in C++.

OF-PENDA takes the mathematical model of an LPN as input, i.e., the matrices Pre, $Post$, M_0, the event-mapping matrix [13] and the fault partitions. While analyzing (K-)diagnosability, the generated model for the K^{th} step are used for $(K+1)^{th}$ step. If the system is diagnosable, the minimum value K_{min} ensuring K-diagnosability is given, and the on-line diagnoser is generated.

In order to show the efficiency of on-the-fly and incremental techniques in diagnosability analysis, the WODES benchmark [6] and the LC diagnosis benchmark are used in the comparative simulation using OF-PENDA and the UMDES library [9], with the help of TINA tool [2], as will be presented in Section 5 and 6. Based on the simulation results, we will point out the similarities and differences between our approach and some existing diagnosis approaches.

5 Application to the WODES Diagnosis Benchmark

5.1 WODES Benchmark

The WODES diagnosis benchmark [6] is shown in Figure 1 and describes a manufacturing system characterized by three parameters: n, m and k, where:

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

The observable transitions are indicated by white boxes, and the unobservable transitions by black boxes.

5.2 Comparative Results

We now analyze the diagnosability upon the fault class $\Sigma_{F1} = \{f_1, \ldots, f_{n-1}\}$. In other terms, only one class of faults is considered here. In order to perform a comparative simulation with OF-PENDA and the UMDES library, some preparations are necessary. The UMDES library deals with automata models by importing a *.fsm* file. Thus, we first generate the reachability graph of the considered PN with the help of TINA yielding to a *.aut* file, which is then transformed into a *.fsm* file by a script integrated in OF-PENDA that we have developed. This ensures that the comparative simulation is performed with the same input.

The simulation has been performed on a PC Intel with a clock of 2.26 GHz and the results are shown in Table 1.

- The first 3 columns entitled "m", "n" and "k" are the basic structural parameters of the WODES diagnosis benchmark.
- The 4^{th} column entitled "$|R|$" is the number of nodes in the marking graph computed by TINA, which is equal to the number of states of the automaton for building the diagnoser by the UMDES library.
- The 5^{th} column entitled "$|\mathcal{N}|$" is the number of the FM-graph nodes.

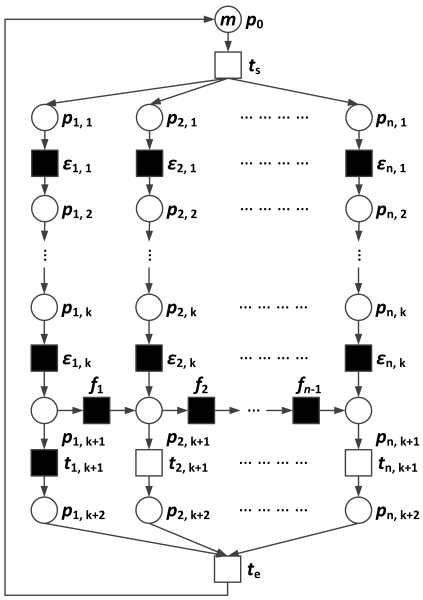

Fig. 1. The WODES diagnosis benchmark

- The 6^{th} column entitled "$|Diag|$" is the number of the diagnoser automaton states generated by UMDES.
- The 7^{th} column entitled "$|\mathcal{X}_v|$" is the number of the fault marking set tree (FM-set tree) nodes.
- The 8^{th} (resp. 9^{th}) column entitled "$\mathcal{D}_D$" (resp. $\mathcal{D}_O$) is the diagnosability verdict returned by UMDES (resp. OF-PENDA), where "Yes" indicates that the system is diagnosable and "No" indicates undiagnosable.
- The 10^{th} column is the stopping value of K in the incremental search. Note that for a diagnosable case (cf. Column 9), this "K" value is equal to "K_{min}".

All the results are obtained under a simulation time of less than 6 hours. "o.t." (out of time) means the result cannot be computed within 6 hours.

5.3 Discussions

A Finer Version of Diagnosability Both the OF-PENDA and the UMDES library allow us to check diagnosability for bounded DESs. In particular, thanks to our incremental investigation of diagnosability, our tool also gives K_{min} for diagnosable systems, while the UMDES library [9] does not. It is worth recalling that the K-diagnosability is a finer version of diagnosability with the following two main features:

Online, the value of K_{min} gives us a valuable piece of information indicating the minimum number of steps necessary to detect and identify faults for a

Table 1. Comparative results obtained by UMDES and OF-PENDA

m	n	k	$\|R\|$	$\|\mathcal{N}\|$	$\|Diag\|$	$\|\mathcal{X}_v\|$	$\mathcal{D}_D$	$\mathcal{D}_O$	K	m	n	k	$\|R\|$	$\|\mathcal{N}\|$	$\|Diag\|$	$\|\mathcal{X}_v\|$	$\mathcal{D}_D$	$\mathcal{D}_O$	K
1	2	1	15	8	4	3	Yes	Yes	2	1	5	1	3,295	2,607	o.t.	66	o.t.	Yes	5
1	2	2	24	10	4	3	Yes	Yes	2	1	5	2	9,691	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.
1	2	3	35	12	4	3	Yes	Yes	2	2	2	1	96	68	20	9	No	No	8
1	2	4	48	14	4	3	Yes	Yes	2	2	2	2	237	137	o.t.	9	o.t.	No	8
1	3	1	80	52	10	6	Yes	Yes	3	2	3	1	1,484	801	20	12	No	No	11
1	3	2	159	90	10	6	Yes	Yes	3	2	3	2	5,949	2,746	o.t.	12	o.t.	No	11
1	3	3	274	138	10	6	Yes	Yes	3	2	4	1	28,203	8,795	o.t.	15	o.t.	No	14
1	3	4	431	196	10	6	Yes	Yes	3	2	4	2	180,918	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.
1	4	1	495	367	29	17	Yes	Yes	4	3	2	1	377	290	66	12	No	No	11
1	4	2	1,200	822	29	17	Yes	Yes	4	3	3	1	12,048	5,165	o.t.	16	o.t.	No	15
1	4	3	2,415	1,533	o.t.	17	o.t.	Yes	4	3	4	1	484,841	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.
1	4	4	4,320	2,554	o.t.	17	o.t.	Yes	4										

diagnosable system. Note that in [1], K is the number of both observable and unobservable transitions after a faulty transition. While here, as well as in [3] K is the value relative to only observable events. Modifying our algorithm to compute both observable and unobservable transitions can be done easily.

Secondly, K-diagnosability allows a meticulous description of the diagnosability for multiple faults. It is advisable to discuss the traditional diagnosability as a series of K-diagnosability problems for each class of faults $\Sigma_{Fi} \subseteq \Sigma_f$. Further analysis on K could help to enhance the diagnosability. In order to solve K-diagnosability for each Σ_{Fi}, it is sufficient to perform our algorithm for each class of faults Σ_{Fi} iteratively. Thus, the computational complexity will be linear with the number of fault classes.

An On-the-fly and Incremental Method The UMDES library deals with diagnosability of systems modeled by automata based on the construction of an observer and a diagnoser automaton, which requires an exhaustive enumeration of the state space. However, we solve the classic diagnosability by handling a series of K-diagnosability problems, where K increases progressively. In other words, we reuse the state space generated while analyzing K-diagnosability to deal with $(K + 1)$-diagnosability. This is totally different from the approach in [1], since for each value of K a new system of equations is generated in their technique based on linear programming. Additionally, by progressively increasing K, it is certain to find K_{min} if the system is diagnosable.

Thanks to the on-the-fly investigation of diagnosability, building the whole FM-set tree is not necessary. Actually, for undiagnosable systems, the FM-set tree building as well as its exploration are stopped as soon as an indeterminate cycle is found. Moreover, for diagnosable systems, while generating a branch in the FM-set tree, we stop as soon as an F-certain or an existing node is obtained. This is a notable advantage compared with the existing approaches [4,

15] which first build an exhaustive diagnoser or a reachability graph. The test result using the WODES diagnosis benchmark has shown this point since we were able to investigate diagnosability on models which are not tractable using UMDES library.

Relation between Diagnosability and Boundedness of PNs The UMDES library solves the diagnosability problem based on exhaustive enumeration of the state space. Therefore, it can only deal with bounded systems modeled by finite state automata.

It is worth noting that there exist two PN-based approaches for the diagnosability analysis of unbounded models. In [3], the authors check the diagnosability of unbounded PN models by analyzing the structure of the generated verifier net reachability graph. This approach requires an exhaustive enumeration of the reachability set of the verifier net, which may be larger than the reachability set of the original PN. In [1], the proposed approach is based on linear programming technique. System behavior is represented by a series of linear equations. The diagnosability of the unbounded PN models can be verified only if the faulty behavior can be described by a finite number of equations.

With the help of the on-the-fly computation, our algorithm can determine undiagnosability as soon as an F_i-indeterminate cycle is detected. Hence, this approach can be applied to some unbounded PNs with an F_i-indeterminate cycle. Although an unbounded PN has infinite fault markings, which may result in an infinite number of fault marking sets (FM-sets), the construction of an FM-set tree terminates once an F_i-indeterminate cycle is detected and a negative diagnosability verdict is emitted. From a practical point of view, some thresholds need to be used if our algorithm is used to deal with unbounded LPNs.

Case of Unlive PNs Note that we have extended our algorithm to also deal with unlive systems, while considering the definition of diagnosability relative to unlive DES, given in [14]. Besides, the WODES diagnosis benchmark we dealt with is unlive when $n \geq 2$, which is against the assumption of liveness in [15]. Concretely, as the computation of the FM-set tree is performed on the fly, we have added an additional stopping condition: when some F-uncertain FM-set containing a deadlock state is obtained. Indeed, in this case, the system may stay indefinitely in this uncertain state, and no diagnosis verdict can be emitted.

Limitations It is shown that some limitations of the WODES benchmark exist when testing diagnosability analysis approaches:

- The benchmark is only live when $k = 1$.
- The benchmark is only diagnosable when $m = 1$.

In order to overcome these limitations, we develop the bounded and live n-line LC benchmark [12], which integrates both diagnosable and undiagnosable faults for any parameter n, as will be introduced in the following section.

6 Application to the LC benchmark

In this section, OF-PENDA is applied to a case study pertinent to railway safety. We study the fault diagnosis problems on an LC system. Our purpose here is two-fold: first, we show that OF-PENDA can deal with complex systems; secondly, the analysis results obtained on a quite complex example can show the efficiency of on-the-fly and incremental techniques in terms of time and memory compared with other existing approaches. Here, the complex model, namely the n-line LC will be modeled by an LPN with a great number of reachable markings.

6.1 n-line LC Benchmark

The n-line LC benchmark, as shown in Figure 2, describes an n-line LC system composed of railway traffic, LC controller and barriers subsystems. The n-line LC benchmark can be obtained from the single-line LC model [10] while fulfilling the following controlling rules under a nominal situation:

- The LC must be closed if any approaching train is detected in any line;
- The LC can be reopened if there is no train in the "within" or "before" sections in any line.

In other terms, the above rules eliminate all the possibilities that the collision between railway and road traffic may take place.

In this global model, all the transitions are observable, but the faulty transitions, i.e., $T_o = T \backslash T_u$ and $T_u = T_f = \{t_6\} \cup (\cup_i \{t_{i,5}\})$.

The n-line LPN model can be rather big when n takes great values. The state space of the corresponding LPN models for the various values of n can be calculated by the TINA tool [2], as will be shown in Table 2.

Recall here that not the whole state space will be generated while using our on-the-fly technique. However, the reachability graphs are generated in order to transform them into the input file (language equivalent automata) for UMDES.

As shown in Table 2, the size of the reachability graph grows very quickly as n increases, since places p_2 and p_6 can hold as many as n tokens, due to which so many markings exist.

6.2 Diagnosability Analysis

In this section, we will analyze the diagnosability of the LC model while considering various values of n. Here, we will consider two fault classes:

- For $T_{F1} = \cup_{i=1}^{n} \{t_{i,5}\}$, we consider all the faults $t_{i,5}$ in each track as belonging to the same class. Recall that such faults express the fact that trains can enter the crossing zone while the barriers are not ensured to be down.
- For $T_{F2} = \{t_6\}$, this fault class depicts an early lowering of the barriers, i.e., before all the trains are ensured to have left the LC.

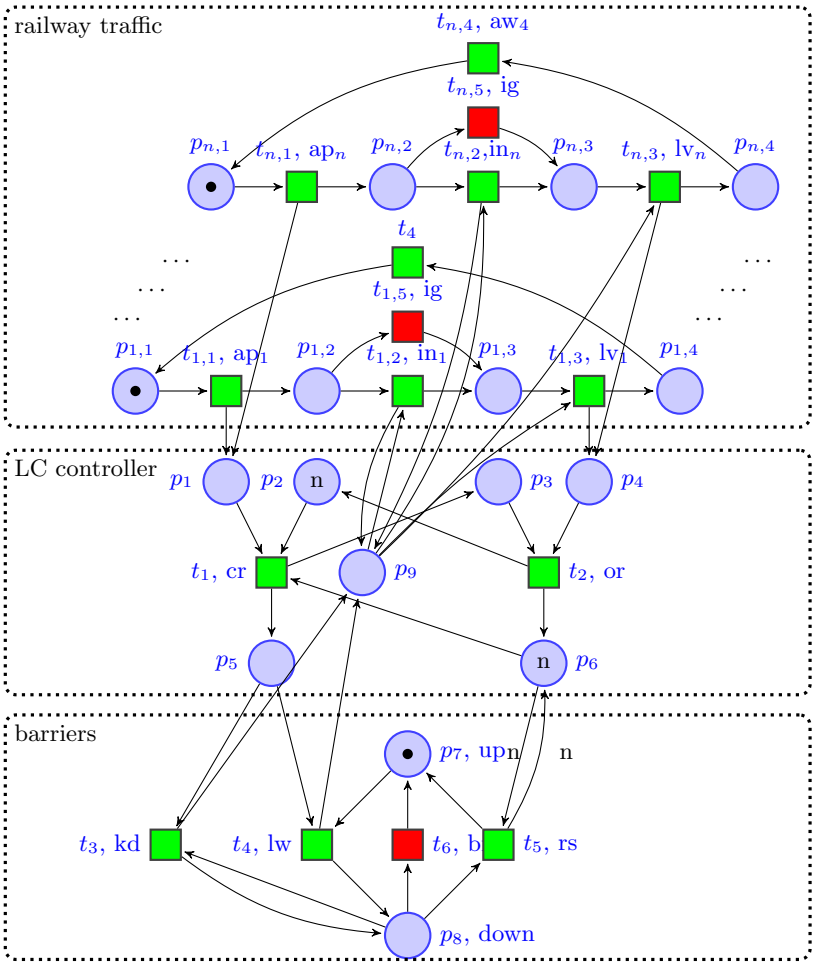

Fig. 2. n-line LC benchmark

A comparative simulation with UMDES library is performed on an Intel PC (CPU: 2.50 GHz, RAM: 3.16 GB) and the results are given in Table 2, Figure 4 and Figure 3, where:

- n is the number of railway tracks;
- Σ_{Fi} is the considered fault class (Σ_{F1} or Σ_{F2});
- $|P|$ and $|T|$ are the number of places and transitions of the LPN model respectively;
- $|A|$ and $|R|$, which are the number of the arcs and the nodes of the reachability graph respectively, give the scale of the LPN reachability graph computed by TINA and which serves as the input automaton states for UMDES;

Table 2. Comparative simulation results based on n-line LC benchmark

n	Σ_{Fi}	$\lvert P\rvert$	$\lvert T\rvert$	$\lvert A\rvert$	$\lvert R\rvert$	$\lvert \mathcal{N}\rvert$	$\lvert Diag\rvert$	$\lvert \mathcal{X}_v\rvert$	$\mathcal{D}_D$	$\mathcal{D}_V$	$\mathcal{D}_O$	K	$\mathcal{T}_T$	$\mathcal{T}_D$	$\mathcal{T}_V$	$\mathcal{T}_O$
1	Σ_{F1}	13	11	28	24	24	26	15	YES	YES	YES	3	<1s	<1s	<1s	<1ms
2		17	16	540	216	116	262	18	a.q.	NO	NO	11	<1s	11s	<1s	15ms
3		21	21	6,256	1,632	173	1,924	18	a.q.	NO	NO	11	<1s	12s	7s	46ms
4		25	26	56,704	11,008	230	12,504	18	a.q.	NO	NO	11	<1s	32s	21m22s	62ms
5		29	31	442,880	68,608	287	75,722	18	a.q.	o.t.	NO	11	2s	28m34s	o.t.	78ms
6		33	36	3,126,272	403,456	344	a.q.	18	o.t.	o.t.	NO	11	11s	1h36m	o.t.	125ms
7		37	41	20,500,480	2,269,184	401	o.t.	18	o.t.	o.t.	NO	11	140s	o.t.	o.t.	156ms
8		41	46	127,074,304	12,320,768	458	o.t.	18	o.t.	o.t.	NO	11	29m	o.t.	o.t.	203ms
9		45	51	o.m.	o.m.	515	-	18	-	-	NO	11	o.m.	-	-	249ms
10		49	56	o.m.	o.m.	572	-	18	-	-	NO	11	o.m.	-	-	296ms
20		89	106	o.m.	o.m.	1,142	-	18	-	-	NO	11	o.m.	-	-	1,467ms
40		169	206	o.m.	o.m.	2,282	-	18	-	-	NO	11	o.m.	-	-	5,460ms
1	Σ_{F2}	13	11	28	24	29	26	15	YES	YES	YES	7	<1s	<1s	<1s	<1ms
2		17	16	540	216	277	262	207	YES	YES	YES	10	<1s	<1s	<1s	453ms
3		21	21	6,256	1,632	2,109	1,924	1842	YES	YES	YES	17	<1s	<1s	5s	109s
4		25	26	56,704	11,008	13,353	12,504	5670	YES	YES	a.q.	18	<1s	20s	16m19s	2h4m
5		29	31	442,880	68,608	o.t.	75,722	o.t.	YES	o.t.	o.t.	o.t.	2s	27m12s	o.t.	o.t.
6		33	36	3,126,272	403,456	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	11s	o.t.	o.t.	o.t.
7		37	41	20,500,480	2,269,184	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	140s	o.t.	o.t.	o.t.
8		41	46	127,074,304	12,320,768	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	29m	o.t.	o.t.	o.t.
9		45	51	o.m.	o.m.	o.t.	-	o.t.	-	-	o.t.	o.t.	o.m.	-	-	o.t.

Note: o.m. = out of memory o.t. = out of time (4 6h) a.q.= accidental quit

results obtained by TINA results obtained by *diag_UR.exe* and *dcycle.exe* of UMDES

results obtained by *verifier_dia.exe* of UMDES results obtained by OF-PENDA

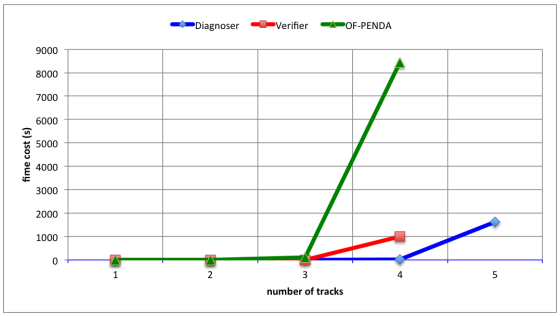

Fig. 3. Time cost for the diagnosability analysis on Σ_{F2}

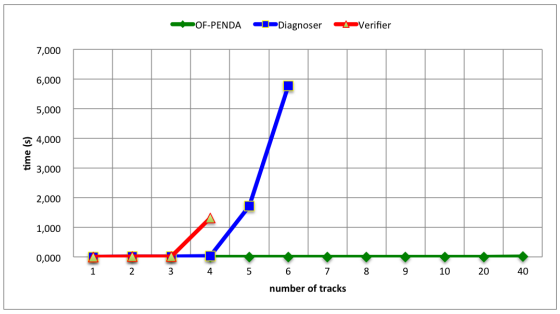

Fig. 4. Time cost for the diagnosability analysis on Σ_{F1}

- $|\mathcal{N}|$ is the number of (on-the-fly) generated fault markings by OF-PENDA when the diagnosability verdict is issued;
- $|Diag|$ is the number of diagnoser states generated by UMDES (the diagnoser approach) which were needed to give the diagnosability verdict;
- $|\mathcal{X}_v|$ is the number of (on-the-fly) generated FM-sets by OF-PENDA when the diagnosability verdict is issued, and corresponds also to the number of nodes of the diagnoser derived from this FM-set tree when the model is diagnosable;
- $\mathcal{D}_D$, $\mathcal{D}_V$ and $\mathcal{D}_O$ are the diagnosability verdicts obtained by diagnoser approach, verifier approach of UMDES and OF-PENDA respectively, where "Yes" indicates that the system is diagnosable and "No" indicates undiagnosable;
- K is the minimum value ensuring diagnosability computed by OF-PENDA, if the system is diagnosable; otherwise, it is the last value under which K-diagnosability is investigated before concluding that the system is undiagnosable;

- $\mathcal{T}_T$ is the time needed to generate the LPN reachability graph computed by TINA, i.e., the time used for preparing the input automaton needed for UMDES;
- $\mathcal{T}_D$, $\mathcal{T}_V$ and $\mathcal{T}_O$ are the times needed to obtain the diagnosability verdict by *dcycle.exe* (diagnoser approach), *verifer_dia.exe* (verifier approach) of UMDES and OF-PENDA (on-the-fly approach), respectively.

6.3 Discussions

For the comparative simulation results, the following remarks can be made:

1. The UMDES library has been integrated in the DESUMA framework which also integrates GIDDES for graphical facilities. Here, we directly use the command lines in UMDES library rather than the interface framework (DESUMA), since the DESUMA takes so much time to load large automaton models.
2. The scale of the PN reachability graph grows quickly (cf. the columns entitled with $|A|$ and $|R|$ in Table 2) as n increases. The capability of TINA to calculate the reachability graph considerably depends on the memory of the computer, since TINA needs to refer to the already built part of the reachability graph all along the computation. One can observe that the generated *.aut* files become considerable starting from $n = 7$. For instance, the size of the *.aut* files for 1, 2, 3, 4, 5, 6, 7-line LC benchmark is 655b, 11KB, 128KB, 1.2MB, 9.9MB, 74.8MB, 518.6MB, respectively. A Mac with a RAM of 16GB cannot store the *.aut* file for 8-line benchmark even if it can be calculated eventually. For the case of $n = 9$, the calculation stops with an "out of memory" error. Thus, the analysis of the cases with $n \geq 9$ cannot be performed using UMDES.
3. The simulation using UMDES and OF-PENDA is performed on a Windows PC. For Σ_{F1}, some "accidental quits" happen during the running of *dcycle.exe* (diagnoser approach in UMDES) for some cases. However, the diagnosability verdict can be obtained by using the verifier technique (command *verifier_dia.exe*) in UMDES library. Note that the relative results given by OF-PENDA when an accidental quit happens are the last outputs before the program's exit. We do not know exactly the reason for this problem. The improvement of the source code and using other operation systems may eliminate the problems.
4. In order to compare the efficiency in terms of time, we make a record of the computing time for each case. The time indicated in the columns entitled with T_o, $\mathcal{T}_D$ and $\mathcal{T}_V$ are obtained by an external timer, since TINA and UMDES do not output this information. Thus, there is an error margin of $\pm$1s. For the OF-PENDA, an automatic timer has been integrated with an error margin of less than 1ms.
5. Besides the diagnosability verdict, OF-PENDA also gives the minimum value of K to ensure diagnosability, which cannot be obtained by UMDES.
6. In these cases, the number of FM-sets generated by OF-PENDA is lower than the number of diagnoser states given by UMDES. This shows the advantage

offered by our on-the-fly technique in terms of memory consumption in terms of memory consumption. Besides, it is worthwhile to mention that the FM-sets can be far fewer than the diagnoser states when an indeterminate cycle exists, i.e., if the model is undiagnosable, and is detected early.

7. It is worthwhile to notice that the FM-set tree is generated by OF-PENDA directly from the LPN, while the UMDES takes an automaton as an input which is equivalent to the reachability graph of the LPN. In other words, the inputs for UMDES and OF-PENDA are not the same and we had to compute the LPN reachability graph *a priori* before performing analysis via UMDES, whereas, on the contrary, the reachable markings are computed on the fly while investigating diagnosability by OF-PENDA.
8. The results (cf. column $\mathcal{T}_D$ and $\mathcal{T}_V$) show that the diagnoser approach is more efficient than the verifier approach while dealing with the n-line LC benchmark. This does not violate the claim that the verifier approach is more efficient in terms of time complexity (polynomial for the verifier approach vs. exponential for the diagnoser approach), since the theoretical complexity is always computed while considering the worst case.

More importantly, compared with the diagnoser and verifier approaches, our on-the-fly approach can deal with some quite complex models that UMDES cannot deal with (e.g., for the cases Σ_{F1} when $n \geq 7$), especially for some undiagnosable systems, and shows better efficiency in terms of time and memory. For example, the diagnosability analysis for the LC model is performed in less than 6 seconds, for even when $n = 40$, whereas UMDES (diagnoser and verifier techniques) do not issue a verdict within 6 hours for $n > 4$. However, for the diagnosable cases (Σ_{F2}), we spend less memory but more time than UMDES, although the obtained results remain comparable: OF-PENDA and UMDES-verifier block from $n = 4$, whereas UMDES-diagnoser blocks at $n = 5$. This gap in terms of efficiency depending on whether the model is diagnosable or not can be explained as follows: For the undiagnosable models, the analysis by OF-PENDA is completed as soon as an indeterminate cycle is found which can occur quickly, hence avoiding the generation and analysis of an important part of the state space. However, for diagnosable models, it is likely that a bigger part of the state space is generated/analyzed since, in this case, the only stopping condition which allows us to avoid building/investigating the whole state space is when a faulty node is generated. Indeed, since we deal with permanent faults, it is useless to continue investigating the following nodes since they are faulty as well.

7 Conclusion

OF-PENDA is a software tool for the diagnosis of DESs modeled by LPNs, on the basis of the techniques developed in [11]. For diagnosable bounded systems, the minimum value K_{min} ensuring K-diagnosability is given. Besides, thanks to the incremental search it implements, OF-PENDA can also issue diagnosis verdict for some unlive and/or unbounded LPN models under some conditions.

Two benchmarks have been used in this paper for illustrating the efficiency of OF-PENDA while a comparison discussion is carried out w.r.t some existing techniques. As future works, we intend to improve the source code of our tool and to integrate some heuristics to guide the depth-first search technique adopted here, while defining some priority rules as regards the branches to be investigated first.

Acknowledgement

The present research work has been partially supported by the International Campus on Safety and Intermodality in Transportation, the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

References

1. F. Basile, P. Chiacchio, and G. De Tommasi. On K-diagnosability of Petri nets via integer linear programming. *Automatica*, 48(9):2047–2058, Sept. 2012.
2. B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA - Construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, July 2004.
3. M. P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. A new approach for diagnosability analysis of Petri nets using verifier nets. *IEEE Transactions on Automatic Control*, 57(12):3104–3117, Dec. 2012.
4. M. P. Cabasino, A. Giua, and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539, Sept. 2010.
5. C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Springer, 2 edition, 2007.
6. A. Giua. A benchmark for diagnosis. http://www.diee.unica.it/giua/WODES/WODES08/media/benchmark_diagnosis.pdf, 2007.
7. 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.
8. S. Koenig, M. Likhachev, Y. Liu, and D. Furcy. Incremental heuristic search in AI. *AI Magazine*, 25(2):99–112, 2004.
9. S. Lafortune. UMDES-Lib software library. http://www.eecs.umich.edu/umdes/toolboxes.html, 2000.
10. N. G. Leveson and J. Stolzy. Analyzing safety and fault tolerance using time petri nets. *Formal Methods and Software Development*, 186, 1985.
11. B. Liu. *An efficient approach for diagnosability and diagnosis of DES based on labeled Petri nets - untimed and timed contexts*. PhD thesis, Univ. Lille Nord de France, 2014.
12. B. Liu. N-track level crossing benchmark. http://6814.is-programmer.com/posts/45619.html, 2014.
13. B. Liu, M. Ghazel, and A. Toguyéni. Toward an efficient approach for diagnosability analysis of DES modeled by labeled Petri nets. In *The 13th European Control Conference (ECC'14)*, 2014.

14. M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929, July 1998.
15. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.
16. S. Schwoon and J. Esparza. *A note on on-the-fly verification algorithms*. Springer, 2005.
17. T.-S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495, Sept. 2002.