A Comparison Between Two Diagnostic Tools Based on Automata and Petri Nets

Stefano Lai, Davide Nessi, Maria Paola Cabasino, Alessandro Giua, Carla Seatzu

Abstract

In this paper we consider two diagnosis procedures for discrete event systems based respectively on automata and Petri nets. First we compare them in terms of applicability and generality. Secondly, we apply them to the WODES diagnosis benchmark and compare them in terms of computational complexity. As a result we conclude that the automata procedure is more general, but the Petri net approach presents significant advantages in terms of computational complexity.

Published as:
S. Lai, D. Nessi, M.P. Cabasino, A. Giua, C. Seatzu, ”A comparison between two diagnostic tools based on automata and Petri nets,” 9th Int. Workshop on Discrete Event Systems (Gteborg, Sweden), pp. 144-149, May 2008.

S. Lai, D. Nessi, M.P. Cabasino, A. Giua and C. Seatzu are with the Department of Electrical and Electronic Engineering, University of Cagliari, Piazza D'Armi, 09123 Cagliari, Italy `stefyial84@libero.it`, `ilnessi@hotmail.com`, `(cabasino,giua,seatzu)@diee.unica.it`.

I. Introduction

The diagnosis of discrete event systems is a research area that has received a lot of attention in the last years and has been motivated by the practical need of ensuring the correct and safe functioning of large complex systems.

Several original theoretical approaches have been proposed using automata, e.g., by Boel and van Schuppen [1], by Debouk *et al.* [2], by Hashtrudi Zad *et al.* [3], by Jiang and Kumar [4], by Lunze and Schroder [5], by Sampath et al. [6], [7], and by Hashtrudi *et al.* [8].

Petri net models have also often been used in this context: the intrinsically distributed nature of Petri nets where the notion of state (i.e., marking) and action (i.e., transition) is local has often been an asset to reduce the computational complexity involved in solving a diagnosis problem. Among the different contributions in this area we recall the work of Ushio *et al.* [9], Benveniste *et al.* [10], [11], Haar *et al.* [12], Jiroveanu and Boel [13], Jiroveanu *et al.* [14], Giua and Seatzu [15], and Cabasino *et al.* [16].

In this paper we focus our attention on two different diagnosis procedures. The first one is based on automata and has been firstly introduced in [7] by Sampath *et al.*. The second one is based on Petri nets and has been proposed by some of us in [15], [16].

In Section III we recall the basic ideas behind such approaches and discuss the differences between them from a theoretical point of view. Although the automata approach only deals with finite state systems, it can be applied to arbitrarily labeled automata and can be used not only to solve the diagnosis problem, but also to check diagnosability, i.e., to establish a priori if a certain fault can be detected after the observation of words of finite length. On the contrary, the Petri net approach allows one to deal also with some classes of infinite state systems, but so far has only been applied to free labeled nets and no results concerning diagnosability are known.

In Section IV we present the WODES diagnosis benchmark, and in Section V we study the computational complexity of diagnosing the benchmark with a tool we developed in Matlab, that implements the Petri net procedure, and the UMDES library [17] that implements the automata procedure. From the results we have collected, we conclude that the Petri net tool is more efficient from a computational point of view. This is not surprising because the Petri net approach is based on the notion of basis markings and j-vectors that allow a compact representation of the reachable state space, while the automata approach is based on the diagnoser that requires an exhaustive enumeration of the state space.

II. Background

A. Finite state automata

A *deterministic finite state automaton* (FSA) is a fourtuple $G = (X, E, \delta, x_0)$ where X is the finite state space, E is the set of events, $\delta : X \times E \longrightarrow X$ is the partial transition function, x_0 is the initial state.

A *nondeterministic finite state automaton* (NFSA) is a fourtuple $G = (X, E, \Delta, x_0)$ where $\Delta \subseteq X \times E_\varepsilon \times X$ is the transition relation, $E_\varepsilon = E \cup \{\epsilon\}$ and ε is the empty trace, i.e., the trace containing no events.

Finally, the language generated by G is $\mathcal{L}(G) = \{w \in E^* : \delta(x, w)$ is defined$\}$.

B. Petri nets

A *Place/Transition net* (P/T net) is a structure $N = (P, T, Pre, Post)$, where P is a set of m places; T is a set of n transitions; $Pre : P \times T \rightarrow \mathbb{N}$ and $Post : P \times T \rightarrow \mathbb{N}$ are the *pre*– and *post*– incidence functions that specify the arcs; $C = Post - Pre$ is the incidence matrix.

A *marking* is a vector $M : P \to \mathbb{N}$ that assigns to each place of a P/T net a non–negative integer number of tokens, represented by black dots. We denote $M(p)$ the marking of place p. A P/T *system* or *net system* $\langle N, M_0 \rangle$ is a net N with an initial marking M_0.

A transition t is enabled at M iff $M \geq Pre(\cdot\,, t)$ and may fire yielding the marking $M' = M + C(\cdot\,, t)$. We write $M\ [\sigma\rangle$ to denote that the sequence of transitions $\sigma = t_{j_1} \cdots t_{j_k}$ is enabled at M, and we write $M\ [\sigma\rangle\ M'$ to denote that the firing of σ yields M'.

Given a sequence $\sigma \in T^*$, we call $\pi : T^* \to \mathbb{N}^n$ the function that associates to σ a vector $y \in \mathbb{N}^n$, named the *firing vector* of σ. In particular, $y = \pi(\sigma)$ is such that $y(t) = k$ if the transition t is contained k times in σ.

A marking M is *reachable* in $\langle N, M_0 \rangle$ iff there exists a firing sequence σ such that $M_0\ [\sigma\rangle\ M$. The set of all markings reachable from M_0 defines the *reachability set* of $\langle N, M_0 \rangle$ and is denoted $R(N, M_0)$. The language $L(N, M_0)$ of $\langle N, M_0 \rangle$ is the set of its firing sequences, i.e., $L(N, M_0) = \{\sigma \in T^* \ \mid\ M_0[\sigma\rangle\}$.

A Petri net having no directed circuits is called *acyclic*.

III. Diagnosis

In this section we briefly recall the diagnosis approach using automata first presented by Sampath *et al.* in [7]. For more details on this we address to [18]. Then, we introduce the diagnosis approach using Petri nets [15], [16]. Finally, we compare the two diagnosis procedures.

A. Diagnosis using automata

The system to be diagnosed is modeled by a NFSA. Faults are modeled by unobservable events, but there may also exist other events that represent legal behaviors that are unobservable as well. Thus we assume that the set of events can be partitioned as $E = E_o \cup E_u$, where E_o is the set of observable events, and E_u is the set of unobservable events. The set of fault events is denoted E_f and it holds $E_f \subseteq E_u$. The set E_f can be further partitioned into r different subsets E_f^i, where $i = 1, \ldots, r$, that model the different fault classes.

Given a NFSA $G = (X, E, \Delta, x_0)$ it is always possible to define a DFSA, called *observer* of G, $Obs(G)$. The projection on E_o of the language generated by G coincides with the language generated by the observer $Obs(G)$, namely with $\mathcal{L}(G)$. Each state of $Obs(G)$ is composed by a subset of X. In particular, the initial state of the observer $x_{0_{obs}}$ includes x_0 and all the states of G that are reachable starting from x_0 executing one or more unobservable events in G. Analogously, a generic state reachable from $x_{0_{obs}}$ with an observable sequence of events $w \in E_o^*$ is composed by the set of states that are reachable in G from the states in $x_{0_{obs}}$ executing w interleaved with zero or more unobservable events.

The *diagnoser* $Diag(G)$ is a DFSA as well and it can be built with the same procedure used for the observer. The states of $Diag(G)$ are composed by ordered sets of $r + 1$ entries, $\{x, l_1, l_2, \ldots, l_r\}$, where $x \in X$ is a state of G and $l_i \in \{F_i, N_i\}$ is the label associated to the ith fault class. In particular, $l_i = F_i$ if x has been reached firing at least one fault event of class i, otherwise $l_i = N_i$.

Note that the state of a diagnoser may contain two different sets relative to the same state $x \in X$ and the same fault class, namely $\{x, l_1, \ldots, l_i, \ldots, l_r\}$ and $\{x, l_1, \ldots, l_i', \ldots, l_r\}$ where $l_i = F_i$ and $l_i' = N_i$. This happens when the same state x can be reached in G executing words that contain fault events in the ith class and words not containing them.

The states of $Diag(G)$ can be classified as follows.

- *Positive state to the ith class*: all entries relative to the ith class are equal to F_i. This means that if the word that leads to such a state is observed, then we can be sure that a fault event in the ith class has occurred.

- *Negative state to the ith class*: all entries relative to the ith class are equal to N_i. This means that if the word that leads to such a state is observed, then we can be sure that no fault event in the ith class has occurred.
- *Uncertain state to the ith class*: it includes both sets labeled with N_i and sets labeled with F_i. In such a case a fault event may either have occurred or not.

The diagnoser can be used to solve two different types of problems.

- *Diagnosis*: given a sequence of observable events w it allows to determine, for each class, if a fault in that class has occurred for sure or not, or it may have occurred, but we cannot be sure of this.
  This is equivalent to establish if the state of the diagnoser that is reached executing w is respectively, positive, negative or uncertain to each class.
- *Diagnosability*: it allows to determine if a system is *diagnosable*, i.e., if it is possible to reconstruct the occurrence of fault events observing words of finite length.

B. Diagnosis using Petri nets

Faults are modeled by unobservable transitions, but there may also exist other transitions that represent legal behaviors that are unobservable as well. Thus we assume that the set of transitions can be partitioned as $T = T_o \cup T_u$, where T_o is the set of observable transitions, and T_u is the set of unobservable transitions. The set of fault transitions is denoted T_f and it holds $T_f \subseteq T_u$. The set T_f can be partitioned into r different subsets T_f^i, where $i = 1, \ldots, r$, that model the different fault classes.

The diagnosis with Petri nets is based on two main notions: *j-vector (or justification-vector)* and *basis marking* [15].

Given an observation $w \in T_o^*$, we denote as $\mathcal{J}(w)$ the *set of justifications*, i.e., the set of *minimal* sequences of unobservable transitions interleaved with w and whose firing enables w. We denote as *j-vectors* the firing vectors relative to the justifications in $\mathcal{J}(w)$.

Finally, the *set of basis markings* $\mathcal{M}_{b,w}$ is the set of markings reached from M_0 firing w interleaved with a justification $\sigma_u \in \mathcal{J}(w)$. The generic marking in $\mathcal{M}_{b,w}$ is denoted $M_{b,w}$.

The diagnoser is a function $\Delta : T_o^* \times \{T_f^1, \ldots, T_f^r\} \longrightarrow \{0, 1, 2, 3\}$ that associates to each observation w and to each fault class T_{f_i}, $i = 1, \ldots, r$, a *diagnosis state*.

- 0: the ith fault cannot have occurred because none of the firing sequences consistent with the observation contains fault transitions of class i.
- 1: a fault transition of class i may have fired but is not contained in any justification of w.
- 2: a fault transition of class i is contained in one (but not in all) justifications of w.
- 3: the ith fault must have occurred, because all firable sequences consistent with the observation contain at least one fault transition of class i.

In [16] we shown that the procedure to evaluate the diagnosis states may be carried out by simply performing matrices multiplications and evaluating the feasibility of certain integer constraint sets. Such a procedure may be applied to all net systems whose unobservable subnet is acyclic.

Clearly, the most burdensome part of the proposed procedure consists in evaluating the feasibility of a finite number of integer constraint sets. However, as shown in [16], in the case of bounded net systems this computation may be moved off-line. An oriented graph, that we call *basis reachability graph* (BRG) may be constructed off-line, that summarizes all the information required for diagnosis. Then, given any observable word w, for any fault class T_f^i, we may evaluate the corresponding diagnosis state by simply looking at the BRG.

The main limitation of this approach is that it only allows one to solve a diagnosis problem. How to use it to also check diagnosability is still an open-issue.

C. Comparison between the two approaches

In this section we compare the above two procedures in terms of applicability and generality of the results. A comparison in terms of computational complexity is proposed in Section V. For more details we address to [19].

The main differences can be summarized as follows.

- The automata approach allows one to check diagnosability while the Petri net one does not.
- The automata approach can be applied to arbitrarily labeled NFSA. On the contrary, the Petri net approach, in its present formulation, is limited to free-labeled nets. However, we believe that it can be easily extended to arbitrarily labeled Petri nets: in such a case a larger number of basis markings should be taken into account.
- Both approaches pose some structural conditions on the unobservable systems. The automata approach requires that no cycle of unobservable events may happen after the occurrence of fault events. The Petri net approach requires a stronger assumption: the unobservable subnet must be acyclic.
- The automata approach is based on NDFA, thus it can only deal with systems with finite state space. On the contrary, the Petri net approach allows one to also deal with unbounded systems, namely systems with an infinite state space.
  Note, however, that the BRG of a Petri net can only be computed if the system has a finite number of basis markings; this condition may fail if the net is unbounded.
- An advantage of the Petri net approach is that looking at the j-vectors, we know not only if the fault has occurred, but we also know how many times it has occurred.

IV. The considered benchmark

The benchmark describes a family of manufacturing systems characterized by three parameters: n, m and k.

- 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.

To obtain one unit of final product n orders are sent, one to each line; this is represented by observable event t_s. Each line will produce a part (all parts are identical) and put it in its final buffer. An assembly station will take one part from each buffer (observable event t_e) to produce the final product.

The part in line i ($i = 1, \ldots, n$) undergoes a series of k operations, represented by unobservable events $\varepsilon_{i,1}, \varepsilon_{i,2}, \cdots, \varepsilon_{i,k}$.

After this series of operations two events are possible: either the part is regularly put in the final buffer of the line, or a fault may occur.

- Putting the part in the final buffer of line 1 corresponds to unobservable event $\varepsilon_{1,k+1}$, while putting the part in the final buffer of line i ($i = 2, \ldots, n$) corresponds to observable event $t_{i,k+1}$.
- There are $n - 1$ faults, represented by unobservable events f_i ($i = 1, \ldots, n-1$). Fault f_i moves a part from line i to line $i + 1$. Note that on line i ($i = 1, \ldots, n-1$) the fault may only occur when the part has finished processing and is ready to be put in its final buffer; the part goes to the same processing stage in line $i + 1$.

A Petri net model of this system is shown in Fig. 1, where thick transitions represent observable event and thin transitions represent unobservable events.

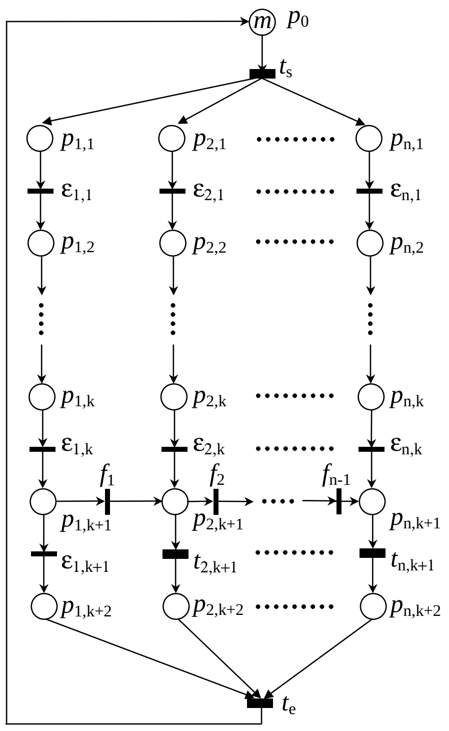

Fig. 1. The considered benchmark.

V. Numerical simulations

In this section we compare the tool we developed in Matlab [20], that implements the Petri net procedure, and the UMDES library [17] that implements the automata procedure. In particular, such a comparison is carried out on the benchmark introduced in Section IV, whose Petri net model is sketched in Fig. 1, while the automaton model corresponds to the reachability graph of the Petri net system that is not reported here for sake of brevity.

Several numerical simulations have been run for different values of n, k and m, that are summarized in Tables I, II and III.

Note that for sake of simplicity we assumed that all faults belong to the same class.

All simulations have been run on a PC Intel with a clock of 1.80 GHz.

- Columns 1 and 2 show the values of n and k.
- Column 3 shows the number of nodes $|R|$ of the reachability graph, and thus the number of states of the NFSA modeling the system.
- Column 4 shows the time t_R in seconds we spent to compute the reachability graph using a function we developed in Matlab.
- Column 5 shows the number of nodes $|BRG|$ of the BRG.
- Column 6 shows the time t_{BRG} in seconds we spent to compute the BRG using a function we developed in Matlab [20].
- Column 7 shows the number of nodes $|Obs|$ of the observer.
- Column 8 shows the time t_{Obs} in seconds we spent to compute the observer using the UMDES library [17].
- Column 9 shows the number of nodes $|Diag|$ of the diagnoser.

n	k	$\lvert R\rvert$	t_R [sec]	$\lvert BRG\rvert$	t_{BRG} [sec]	$\lvert Obs\rvert$	$t_{Obs}[sec]$	$\lvert Diag\rvert$	t_{Diag} [sec]
2	1	15	0.011	5	¡ 0.03	4	0.053	12	0.048
2	2	24	0.019	5	¡ 0.03	15	0.060	39	0.049
2	3	35	0.026	5	0.032	22	0.059	58	0.050
2	4	48	0.038	5	0.032	34	0.060	88	0.053
3	1	80	0.074	17	0.12	51	0.061	151	0.056
3	2	159	0.234	17	0.145	228	0.063	698	0.084
3	3	274	0.693	17	0.15	494	0.073	1480	0.155
3	4	431	1.85	17	0.15	794	0.12	2405	0.31
4	1	495	2.52	69	0.51	4172	0.619	14009	16.66
4	2	1200	19.04	69	0.57	32132	319.6	126041	4874
4	3	2415	95.33	69	0.67	120083	6169.4	*o.t.*	*o.t.*
4	4	4320	368.8	69	0.766	*o.t.*	*o.t.*	*n.c.*	*n.c.*
5	1	3295	203.6	305	4.82	*o.m.*	*o.m.*	*n.c.*	*n.c.*
5	2	*o.t.*	*o.t.*	305	5.06	*n.c.*	*n.c.*	*n.c.*	*n.c.*
5	3	*o.t.*	*o.t.*	305	6.37	*n.c.*	*n.c.*	*n.c.*	*n.c.*
5	4	*o.t.*	*o.t.*	305	6.6	*n.c.*	*n.c.*	*n.c.*	*n.c.*

TABLE I
NUMERICAL RESULTS IN THE CASE OF m = *1*.

n	k	$\lvert R\rvert$	t_R [sec]	$\lvert BRG\rvert$	t_{BRG} [sec]	$\lvert Obs\rvert$	$t_{Obs}[sec]$	$\lvert Diag\rvert$	t_{Diag} [sec]
2	1	96	0.094	17	0.078	349	0.125	3864	1.764
2	2	237	0.521	17	0.078	9849	9.754	*o.m.*	*o.m.*
3	1	1484	27.19	140	1.17	*o.t.*	*o.t.*	*n.c.*	*n.c.*
3	2	5949	648.8	140	1.031	*o.t.*	*o.t.*	*n.c.*	*n.c.*
4	1	28203	20622	1433	86	*o.t.*	*o.t.*	*n.c.*	*n.c.*
4	2	*o.t.*	*o.t.*	1433	86.4	*n.c.*	*n.c.*	*n.c.*	*n.c.*

TABLE II
NUMERICAL RESULTS IN THE CASE OF m = *2*.

n	k	$\lvert R\rvert$	t_R [sec]	$\lvert BRG\rvert$	t_{BRG} [sec]	$\lvert Obs\rvert$	$t_{Obs}[sec]$	$\lvert Diag\rvert$	t_{Diag} [sec]
2	1	377	1.221	39	0.25	*o.t.*	*o.t.*	*n.c.*	*n.c.*
2	2	1293	19.038	39	0.25	*o.t.*	*o.t.*	*n.c.*	*n.c.*
3	1	12048	2978	553	7.65	*o.t.*	*o.t.*	*n.c.*	*n.c.*
3	2	*o.t.*	*o.t.*	553	7.55	*n.c.*	*n.c.*	*n.c.*	*n.c.*
4	1	*o.t.*	*o.t.*	9835	2392.3	*n.c.*	*n.c.*	*n.c.*	*n.c.*
4	2	*o.t.*	*o.t.*	9835	2395.3	*n.c.*	*n.c.*	*n.c.*	*n.c.*

TABLE III
NUMERICAL RESULTS IN THE CASE OF m = *3*.

- Column 10 shows the time t_{Diag} in seconds we spent to compute the diagnoser using the UMDES library [17].

Note that, since in some cases the times to run simulations are very short, in order to minimize the variance of such times — related to the concurrency of the processes executed by the processor — the average time over several simulations has been computed. In particular, the first 9 rows of Table I and the first row of Table II show the average time over 100 simulations. The 10th row of Table I shows the average time over 20 simulations. In all the other cases only one simulation has been run.

Some boxes of the above tables contain non numerical values.

- $o.t.$ (*out of time*): denotes that the corresponding value has not been computed within 6 hours;
- $n.c.$ (*not computable*): denotes that the corresponding value cannot be computed: e.g., if the observer is *o.t.* the corresponding diagnoser cannot be evaluated;
- $o.m.$ (*out of memory*): denotes that the corresponding value has not been computed because the virtual memory of the calculator has run out.

Tables I, II and III show that the time spent to compute the reachability graph, the observer and the diagnoser highly increases with the dimension of the net, namely with n and k, and with the number of products m. In same cases it has been even impossible to compute them.

On the contrary, the time spent to compute the BRG is always reasonable even for high values of n, k and m.

We can observe that there is not a clear relationship between the number of nodes of the reachability graph and the number of nodes of the observer. As an example, let us consider the two cases: $n = 3$, $k = 3$, $m = 1$ and $n = 2$, $k = 2$, $m = 2$. In the first case the number of nodes of the reachability graph is $|R| = 274$ and the number of nodes of the observer is $|Obs| = 494$. In the second case $|R| = 237 < 274$ and $|Obs| = 9849 >> 494$.

We can also observe that, as expected, the number of states of the diagnoser $|Diag|$ is greater than the number of states of the observer $|Obs|$. Thus, it is always possible to construct the observer if it is possible to construct the diagnoser, but not viz (e.g., $n = 4$, $k = 3$, $m = 1$). However, there is not a clear relationship between the complexity in evaluating them. As an example let us consider the two cases: $n = 4$, $k = 2$, $m = 1$ and $n = 2$, $k = 2$, $m = 2$. In the first case $|Obs| = 32132$ and it has been possible to also compute the diagnoser. In the second case $|Obs| = 9849 << 32132$ and it has not been possible to compute the diagnoser within 6 hours.

Tables I, II and III also show that the number of nodes of the BRG only depends on n and m, while it is invariant with respect to k. Only the contrary, $|R|$, $|Obs|$ and $|Diag|$ also highly increases with k.

The relationship among the time to compute the reachability graph, the BRG, the observer and the diagnoser, respectively, and the values of n and k is better highlighted in Fig. 2 in the case of $m = 1$.

Looking at Fig. 2 it can be noticed that while t_{BRG} slowly increases with n and k, t_R, t_{Obs} and t_{Diag} highly depend on these parameters. Moreover, while the BRG is computable for all considered values of n and k, the reachability graph is only computable for $n \leq 4$ if $k \geq 2$; the observer is only computable for $n \leq 3$ if $k = 4$, and for $n \leq 4$ if $k = 1, 2, 3$; finally, the diagnoser is only computable for $n \leq 3$ if $k = 3, 4$, and for $n \leq 4$ if $k = 1, 2$.

On the basis of the above simulations we can conclude that from a computational point of view, the Petri net tool is better than the automata tool. In particular, in several cases we realized that the diagnoser cannot be built at all (at least in a reasonable time), while the BRG can always be computed in a sufficiently small time.

This is not surprising: in fact thanks to the basis markings the reachability space can be described in a compact manner. On the contrary the automata approach is based on an exhaustive enumeration of all reachable states. This advantage is particularly evident in the case of concurrent systems [19].

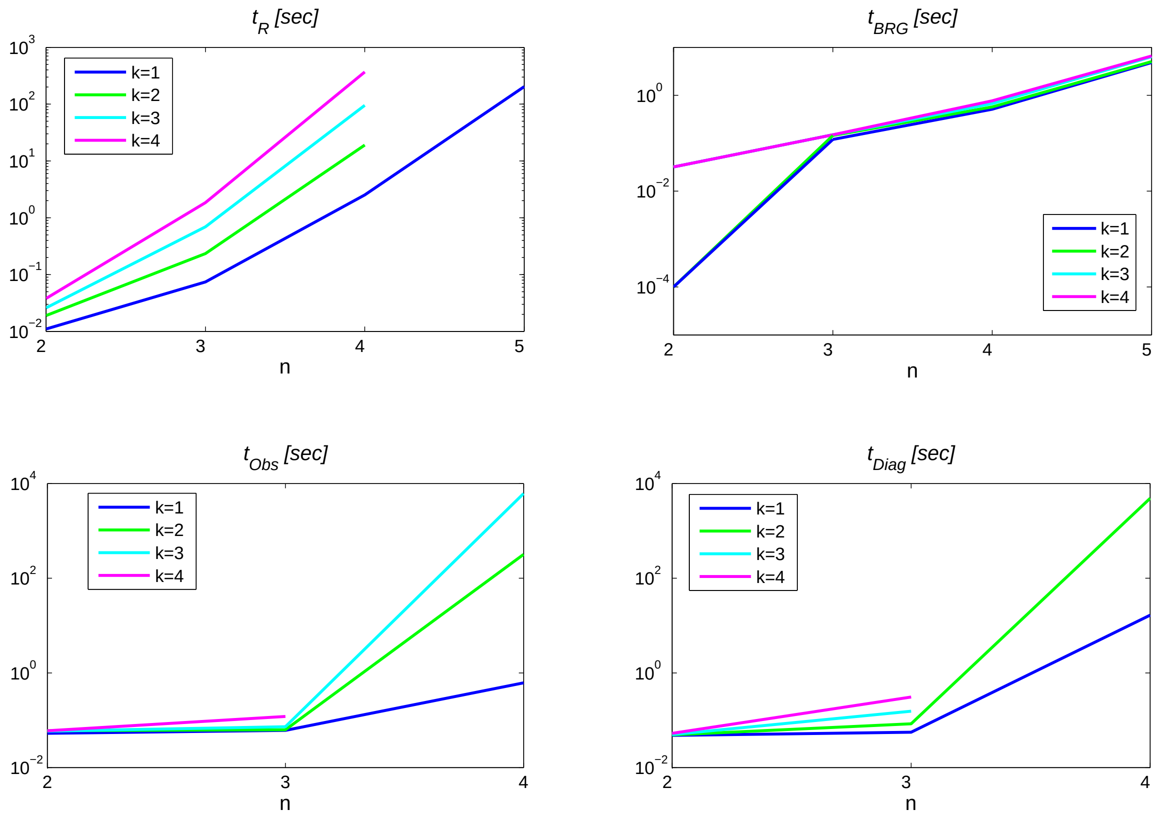

Fig. 2. The computational times t_R, t_{BRG}, t_{Obs} and t_{Diag} with respect to n and k, in the case of $m = 1$.

On the other hand, as already highlighted above, the Petri net approach is only applicable when performing on-line diagnosis, while the automata approach also provides necessary and sufficient conditions for diagnosability.

We can finally observe that the small number of simulations we have been able to carry on using automata does not allow us to evaluate how the computational complexity is related to n and k in the automata approach. However, we can conjecture an exponential dependence on n since the computational time varies from few seconds to "*out of time*" slightly increasing n.

Evaluating the dependence on k is even more difficult: for $m = 1$ and $n = 2$ the times to compute the diagnoser are very small and comparable for all values of k; for $m = 1$ and $n = 3$ the time is still reasonable for all values of k, even if it increases approximately in a linear way with k; for $m = 1$ and $n = 4$ it becomes "*out of time*" for $k = 3$.

Discussion

We have compared the complexity of solving the WODES benchmark problem with two different tools, one using the BRG and the other one using the diagnoser. The diagnoser has size which is exponential in the number of system states, whereas the size of the BRG is linear in the size of the state space, being at most equal to the size of the reachability graph. Thus, it was expected that the second tool should have better performance.

Note however, that it does not necessarily follow that Petri nets diagnosis approaches are better than automata based approaches. We remind that there also exist other automata based approaches that do not require an exhaustive enumeration of the diagnoser states. As an example, in [3], a subgenerator of the Reachability Transistion System (RTS) reachable from the initial state, is used in a similar fashion as the BRG. Since the system states that are only on unobservable trajectories do not show up in the reachable subgenerator of RTS, the size of the reachable subgenerator of RTS is also smaller than the original system. It would be interesting to test the WODES benchmark problem on a tool based on this approach and compare with the results obtained using our tool.

VI. Conclusions

In this paper we compared two diagnosis approaches of discrete event systems. The first one is based on automata and the second one on Petri nets. In particular, we first discussed the differences between them in terms of applicability and generality of the results and showed that the automata approach is more general. Then, we considered a benchmark and compared them in terms of computational complexity: from this point of view the Petri net approach is much more efficient, and allows one to deal with larger dimensional systems.

VII. Acknowledgment

We thank Andres Rey for his help in the development of the tool MATLAB for the construction of the BRG [20].

References

[1] R. Boel and J. van Schuppen, “Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers,” in *Proc. WODES’02: 6th Work. on Discrete Event Systems (Zaragoza, Spain)*, Oct. 2002, pp. 175–181.

[2] R. Debouk, S. Lafortune, and D. Teneketzis, “Coordinated decentralized protocols for failure diagnosis of discrete-event systems,” *Discrete Events Dynamical Systems*, vol. 20, pp. 33–79, 2000.

[3] S. H. Zad, R. Kwong, and W. Wonham, “Fault diagnosis in discrete-event systems: framework and model reduction,” *IEEE Trans. Automatic Control*, vol. 48 (7), pp. 1199–1212, 2003.

[4] S. Jiang and R. Kumar, “Failure diagnosis of discrete-event systems with linear-time temporal logic specifications,” *IEEE Trans. Automatic Control*, vol. 49, no. 6, pp. 934–945, Jun. 2004.

[5] J. Lunze and J. Schroder, “Sensor and actuator fault diagnosis of systems with discrete inputs and outputs,” *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 34, no. 3, pp. 1096–1107, Apr. 2004.

[6] M. Sampath and S. Lafortune, “Active diagnosis of discrete-event systems,” *IEEE Trans. Automatic Control*, vol. 43, pp. 908–929, 1998.

[7] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, “Diagnosability of discrete-event systems,” *IEEE Trans. Automatic Control*, vol. 40 (9), pp. 1555–1575, 1995.

[8] S. H. Zad, R. Kwong, and W. Wonham, “Fault diagnosis in discrete-event systems: framework and model reduction,” *IEEE Trans. Automatic Control*, vol. 48, no. 7, pp. 1199–1212, Jul. 2003.

[9] T. Ushio, L. Onishi, and K. Okuda, “Fault detection based on Petri net models with faulty behaviors,” in *Proc. SMC’98: IEEE Int. Conf. on Systems, Man, and Cybernetics (San Diego, CA, USA)*, Oct. 1998, pp. 113–118.

[10] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, and C. Jard, “Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri nets,” *Discrete Events Dynamical Systems*, vol. 8, pp. 203–231, Jun. 1998.

[11] A. Benveniste, E. Fabre, S. Haar, and C. Jard, “Diagnosis of asynchronous discrete event systems, a net unfolding approach,” *IEEE Trans. Automatic Control*, vol. 48, no. 5, pp. 714–727, May 2003.

[12] S. Haar, A. Benveniste, E. Fabre, and C. Jard, “Partial order diagnosability of discrete event systems using Petri net unfoldings,” in *Proc. 42th IEEE Conf. on Decision and Control*, Dec. 2003, pp. 3748– 3753.

[13] G. Jiroveanu and R. Boel, “A distributed approach for fault detection and diagnosis based on time Petri nets,” *Mathematics and Computers in Simulation*, vol. 70, pp. 287–313, 2006.

[14] G. Jiroveanu, R. Boel, and B. D. Schutter, “Fault diagnosis for time Petri nets,” in *Proc. WODES’06: 8th Work. on Discrete Event Systems (Ann Arbor, Michigan, USA)*, Jul. 2006, pp. 313– 318.

[15] A. Giua and C. Seatzu, “Fault detection for discrete event systems using Petri nets with unobservable transitions,” in *Proc. 44th IEEE Conf. on Decision and Control*, Dec. 2005, pp. 6323–6328.

[16] M. P. Cabasino, A. Giua, and C. Seatzu, “Fault detection for discrete event systems using Petri nets with unobservable transitions,” *Automatica*, (Submitted).
[17] S. Lafortune, “Umdes-lib software library. [online]. available: *http://www.eecs.umich.edu/umdes/toolboxes.html*.”
[18] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems - Second Edition*. Springer, 2007.
[19] M. P. Cabasino, “Diagnosis of discrete event systems using automata and Petri nets,” Master’s thesis, Dep. Electric and Electronic Engineering, University of Cagliari, Cagliari, Italy, 2005. (In Italian).
[20] A. Rey, “Diagnosis of Petri nets using the Basis Reachability Graph,” Master’s thesis, Dep. Electric and Electronic Engineering, University of Cagliari, Cagliari, Italy, 2007. (In Italian).