DREAMAP 2012

Diagnostic REAsoning: Model Analysis and Performance

August, 27th, Montpellier, France

Editors:

Yannick Pencolé, LAAS, CNRS, University of Toulouse, France
Alexander Feldman, University College Cork, Ireland
Alban Grastien, NICTA, Australian National University, Australia

Introduction

Diagnosis is the reasoning process that aims at determining whether certain properties (such as the occurrence of a failure, of an unexpected measurement, or of a deviation from prescribed behavior) hold at a given time inside a system (fault diagnosis) and/or in its environment (situation diagnosis) and identifying their causes. Diagnosis reasoning is thus crucial for many reasons: safety/maintainability in supervised systems, robustness/decision autonomy of smart agents in their partially observed environment, reconfiguration of business strategies after a failure... Despite the huge spectrum of applications, Model-Based Diagnosis (MBD) problems are generic as any diagnosis algorithm relies on a class of models, or a modeling framework, that represents a larger class of systems. In the last decades, many diagnostic modeling frameworks have been proposed that are now well established.

Nowadays, the community is more and more interested in understanding the power and the limits of such frameworks/techniques. Such studies allow to determine in advance how the diagnosis algorithm will behave. This is crucial especially if Diagnosis is the input of other automated reasoning tools (like planning, scheduling,...). The accuracy and performance of MBD algorithms depend to a large extent on some properties of the underlying model (correctness, fidelity, accuracy, diagnosability, predictability...). For example, the size of the model (measured as the number of variables and constraints) affects all major diagnostic metrics such as diagnostic accuracy, computational performance, etc. Tools for model analysis are necessary to assist both in the modeling and in the computation phases of MBD. Model analysis can be used for creating worst-case scenario benchmarks, asserting model correctness, and facilitating automatic or semi-automatic modeling. As such, model-based analysis can be done with the help of a variety of AI and optimization tools such as SAT and Max-SAT methods, search algorithms, optimal sensor placement algorithms, and others.

The purpose of the DREAMAP workshop, held in conjunction with the 20th European Conference on Artificial Intelligence, is to gather researchers in order to fill the gaps and exchange ideas about which reasoning techniques are needed for the analysis of the various models used in MBD and the implication of these analyses to the performance of MBD algorithms.

Enjoy,

The DREAMAP Editors.

Papers

A 3-Valued Logic for Diagnostic Applications

Antoni Ligęza[1]

Abstract. This paper presents a yet another three-valued logic for AI applications. The proposed logic is motivated by strictly technical analysis of systems, and can be applied in domains such as intelligent system monitoring, automated diagnosis, or intelligent control. Contrary to some other 3-values logics developed so far, the one presented here builds on one nominal *true* value and two opposite *false-values* — *false-negative* and *false-positive*. The main technical intuition behind this proposal is that in numerous technical systems negation of correct state can be interpreted as one having either lower or higher value of certain signal. With use of the proposed logic modeling dynamic systems can be more precise and generated diagnoses are more refined.

1 INTRODUCTION

One of the principal investigation issues of AI is to develop *Knowledge Representation* formalisms and automated inference tools, so that complex tasks that require intelligent reasoning can be accomplished. By now, there seems to be a general agreement, that there can be no single, universal solution for *modeling intelligence*.

The plethora of logical formalisms and languages aimed at capturing characteristics such as nonmonotonicity, impreciseness, time, etc. is a spectacular confirmation of failure to cover the functionality of natural language, and especially — its expressive power.

This paper is focused on some *logical perspective* on diagnostic reasoning. Methods of formal description of diagnostic inference are diversified. There are algebraic, graph-based, logical and mathematical model-based diagnostic approaches. Some of the popular models include extended diagnostic matrices [11, 13, 12], set-covering model [20, 21], consistency-based reasoning [22, 9, 17], logical causal graphs [6, 15, 17], and many other [24, 23, 3, 10].

In the area of Fault Detection and Isolation (FDI) emerging from classical Automatic Control mathematical models, such as algebraic and differential equations, are mostly in use [4, 5, 10]. However, other tools, such as causal graphs, fuzzy rule-based systems, and neural networks are in application as well.

A survey and comparison of the approaches coming from the Automatic Control and the FDI Community confronted with the ones based on Artificial Intelligence (AI) techniques and developed by the DX Community[2] is provided in [2, 1]. It turns out that both the approaches, developed almost independently by the FDI and DX researchers, are highly analogous, and there are intrinsic similarities between analytical redundancy analysis and consistency-based reasoning. Moreover, both of the approaches are based on similar assumptions and use only the model of correct behavior of the system. Neither expert nor statistical knowledge, e.g. in the form of evidence of fault history are taken into account.

In general, the main conceptual approaches to building a diagnostic engine can be classified as follows:

- *expert-type* — where expert diagnostic knowledge is encoded in the form of rules, decision trees, decision graphs or decision tables; this includes also case-based reasoning,
- *causal models and abductive reasoning* — where a causal model is available and abductive reasoning is used to put forward diagnostic hypotheses,
- *model-based consistency-based reasoning* — where a model of positive behavior is available and diagnoses are generated through inconsistency elimination while observations are inconsistent with the expected behavior.

Here we are concerned with the third type of approach. Note that, solving a diagnostic problem within this framework can be considered as a kind of *constraint programming* problem [18]; the variables denoting components can be assigned values, such as *ok* and *faulty*, so that consistency is regained.

In this paper an attempt to move the variable assignment to the logical level of different *truth-values* is put forward. The work presents a yet another three-valued logic for AI applications. The proposed logic is motivated by strictly technical analysis of systems, and can be applied in domains such as intelligent system monitoring, automated diagnosis, or intelligent control. Contrary to some other 3-values logics developed so far, the one presented here builds on one nominal *true* value and two opposite *false-values* — *false-negative* and *false-positive*. The main technical intuition behind is that in numerous technical systems false correct state can be interpreted as one having either lower or higher value of certain signal. With use of the proposed logic modeling dynamic systems can be more precise and generated diagnoses are more refined.

The key issue for diagnoses refinement is based on inconsistency detection, analysis and elimination. Inconsistency detection takes places when two chunks of knowledge that cannot be satisfied at the same time appear in the set of deduced consequences and observations. Inconsistency is then eliminated by appropriate selection and refinement of diagnostic hypotheses. All potential diagnoses left must be consistent with all the other knowledge at hand.

Inconsistency can be checked for in a purely logical way (e.g. p and $\neg p$ are present in the knowledge under discourse), or as *material inconsistency*, when two pieces of knowledge are invalid together due to the assumed interpretation.

The proposed formalism is not truth-functional in the sense that for any two defined truth values and for any logical connectives one can determine the resulting truth value. This is also the case of other non-trivial multi-valued logics. For example the Łukaszewicz logic applied in relational databases has the so called NULL (or UNDE-

[1] AGH University of Science and Technology, Krakow, Poland, email: ligeza@agh.edu.pl

[2] DX is a series of conferences devoted to automated diagnosis methods emerging from AI.

TERMINED) value. Here instead of introducing some artificial value for covering uncertainty we prefer to provide a set of still-possible values. The fact that in some diagnostic reasoning cases (e.g. based on abduction) we are not able to precisely define the logical value is an intrinsic feature of such types of reasoning. In abduction one states hypotheses and try to refine them. This will be explained with an example further on.

The material presented in this paper is based in part on some previous works[3]. The concept of *Potential Conflict Structure* was first presented in [14]. It was further developed in [16], [7], and especially in the Ph.D. Thesis [8]. The concept of AND/OR graphs were introduced in [15]. Basic logical material and the Reiter's theory were presented in [17]. The diagnostic procedure presented here and based on modeling the space of diagnostic hypotheses with AND/OR graphs is coming from [19]. The ideas on qualitative three-valued diagnoses have been introduced in [19], and the diagnostic part of the presentation here is based on [18].

2 MULTI-VALUED LOGICS: STATE-OF-THE-ART

First multi-valued logics were introduced as early as in 1920 by Łukasiewicz. From that time a number of proposals have been put forward by Łukasiewicz, Post, Kleene, Priest, Belnap, Jaśkowski, Sobociński, Słupecki, and perhaps many other. A modern practical approach is the simple fuzzy logic with numerous possibilities of T-norms and T-conorms, and some further constructs based on possibility theory and modal logics.

Most of the work done within the area of three-valued logics (also Belnap four-valued one) turn around the concept of one definite truth value and one false value, and some values in-between with semantics of the type *undetermined* or *partially true/false*. In general, this is also the case of simple fuzzy logic. Our proposal seems to be different; perhaps some relationship to qualitative physics of Kuipers can also be observed.

3 MOTIVATION AND INTUITIONS

In this section we present motivations behind the proposed logic. In fact, it is based on simple engineering intuition. The basic assumptions are as follows.

In classical, two-valued logics, purely logical negation can be considered not to be constructive. Instead of negating a logical statement p, which leads to $\neg p$, one can think about replacing $\neg p$ with a constructive statement q. Obviously, the meaning of q must depend on the interpretation (the universe of discourse and mapping of the symbols into relations defined over this universe).

For example, referring to a simple universe with two-state elements, one can say that if $p \equiv (state(component) = on)$, then $\neg p$ can be defined as $state(component) = off$ with the obvious meaning.

We shall refer to this kind of negations as *material negation*. Note that material negation always refers to the universe of discourse and the intended interpretation. It is by nature constructive. It can be very helpful in analysis and understanding of physical systems.

This classical understanding of negation is like operating in a world where every object is just *white* or *black*. If it is *not* white, then it must be black, and vice versa. But if one admits more colors to play, the situation changes, and the statement that something is *not* white is no longer equivalent to claiming that it is black; in fact, its color is *constrained* to belong to some set.

In more complex worlds, there can be no simple one-to-one relationship between logical and material negation. Instead of black-and-white objects world consider one with colors such as *red*, *green*, and *blue*. Now, the material negation of $p \equiv$ (*color(object) = red*) would be equivalent to $\neg p \equiv$ (*color(object) = green* $\vee$ *color(object) = blue*). In such a case one is placed against some ambiguity, since the material negation is no longer defined in a unique way.

Now, in case of technical systems we often observe signals which should be stabilized around some required values (the so-called *set-points*). A statement such as 'it is OK' can refer to a correct value, while its negation to incorrect one. The key idea is that a new, more precise negation can directly point to one of the alternative values, while set-valued negation can define the constraints on negated proposition.

For the purpose of this paper we assume just two *false-values*; the key ideas are as follows:

- there are three defined logical values:
  - 0 — for intuition *true* or normal/nominal state,
  - $-$ — *false-negative* — below the expected state, and
  - $+$ — *false-positive* — above the expected state,
- each proposition (formula) can take only a single logical value at a time,
- a combination of two different logical values can be considered as a *constraint* imposed over a formula.

In fact, in more realistic treatment, a component can fail in several ways; thus, one can speak about *type of fault* or *faulty mode*. It is proposed to distinguish one correct mode, denoted with 0 and two faulty modes, denoted with $+$ and $-$, with the obvious intuitive meaning (above or below the expected value of the parameter).

4 LANGUAGE: NOTATION AND LOGICAL CONNECTIVES

Consider a set of propositional symbols $P = \{p_1, p_2, \ldots, p_n\}$. Further, let $\mathcal{I}$ be an interpretation, i.e. a function assigning each propositional symbol its current *logical value* (if known). Let $\mathcal{T} = \{-, 0, +\}$ be the set of admissible truth values. Hence, $\mathcal{I}$ is a (partial) function of the form: $\mathcal{I}\colon P \to \mathcal{T}$.

In order to simplify the notation, instead of writing $\mathcal{I}(p) = \tau$, where $\tau \in \mathcal{T}$ we shall write $p[\tau]$. Further, we introduce a kind of *constraints* over the logical value of p; by $p[\tau_1, \tau_2]$ we mean that p takes as its logical value τ_1 or τ_2. Note that, this in fact allows to specify some *uncertainty* w.r.t the logical value of p. For example, $p[-, 0]$ means that either $p[-]$ or $p[0]$ holds. Surely, if the value of p is determined, we have $p[-, 0, +]$ which can be considered as an extension of the *excluded middle law* from classical, two-valued logic. Finally, $p[?]$ means that the value of p is undetermined.

It is also convenient to assume that the elements of $\mathcal{T}$ are ordered, and the order is $(-, 0, +)$, i.e. $- < 0 < +$. In a sense, the *true* is located in a central point (for engineering intuitions — a nominal value). The $-$ and $+$ represents deviations from 0. Since we have one *true* value and two *false* values, the shorthand to refer to this logic is 1T-2F.[4]

[3] Of the author and his former Ph.D. student Barłomiej Górny, Ph.D. and some common work with Jan Maciej Kościelny.

[4] One can easily imagine extensions of the form kT-nF with k different true values and n false values; further, extensions to kT-mM-nF with m middle values are possible.

Assume the elements of $\mathcal{T}$ are ordered, and the order is $(-,0,+)$. Let us introduce two simple operators, $p\prime$ — circular shift of the value of p right and $p\backslash$ — circular shift of the value of p left. For example, if $p[0]$, then $p\prime[+]$, $p\prime\prime[-]$, and, finally $p\prime\prime\prime[0]$. The extended version of the law of excluded middle can be now formulated as follows:

$$\models p\backslash \vee p \vee p\prime, \tag{1}$$

or in equivalent way as $\models p \vee p\prime \vee p\prime\prime$ ($\models p \vee p\backslash \vee p\backslash\backslash$).

4.1 Negation

The key issue in the proposed 1T-2F logic refers to modified, constructive understanding of negation. As it was mentioned, there are three definite logical values: *true* (denoted with 0), *false-negative* (denoted with $-$) and *false-positive* (denoted with $+$). The consequence of that is that negation of *true* is not defined in a unique way: it can be both *false-negative* as well as *false-positive*. In general, negation of a valid proposition leads to a set of admissible false values. The negation is in fact understood as set-complement of logical values.

The detailed operation of negation is given in Table 1. Negation of

Table 1. The formal definition of *negation* operator

proposition	negated proposition
$p[0]$	$p[+,-]$
$p[+]$	$p[0,-]$
$p[-]$	$p[+,0]$
$p[-,0]$	$p[+]$
$p[-,+]$	$p[0]$
$p[0,+]$	$p[-]$

a propositional symbol p will be denoted as $\bar{p}$ or $\neg p$; negation of a more complex formula Φ will be denoted as $\neg(\Phi)$.

It is straightforward to observe that the *Principle of Excluded Middle* is no longer true in its basic form. However, its extended version can be put forward as follows:

$$p[0] \vee p[+] \vee p[-] \tag{2}$$

which can be a useful constraint in potential diagnoses elimination procedures.

We introduce also a more refine *directed negation*. Having in mind the physical interpretation of the truth-values, and the order among them, we define one step *negation down* ($\rightharpoondown$) and one-step *negation up* ($\rightharpoonup$) operating as follows: $\rightharpoondown p[+] = p[0]$, $\rightharpoondown p[0] = p[-]$, $\rightharpoondown p[-] = p[-]$; $\rightharpoonup p[-] = p[0]$, $\rightharpoonup p[0] = p[+]$, $\rightharpoonup p[+] = p[+]$. This time the value of negation is defined in a unique way. Moreover, analogous but circular negation can be defined as above, with two additional rules assuring circularity, namely ($\ominus\!\rightharpoondown p[-] = p[+]$ and $\ominus\!\rightharpoonup p[+] = p[-]$).

4.2 Conjunction

Conjunction here means in fact composition of two signals represented by propositions. The following table defines it in a formal way. The conjunction will be denoted with standard symbol $\wedge$; so $p \wedge q$ is the conjunction of p and q. Note that inference may be possible both for single and double truth-values. For example, for $p[-,0]$ and $q[0]$ we can infer $(p \wedge q)[-,0]$; it takes the form of *constraint propagation*. For $p[-,0]$ and $q[-]$ we have $(p \wedge q)[-]$; there is a refinement of the final value. For For $p[-,0]$ and $q[0,+]$ we have $(p \wedge q)[?]$, i.e. the result is totally undefined.

Table 2. Formal definition of conjunction

$\wedge$	-	0	+
-	-	-	?
0	-	0	+
+	?	+	+

5 APPLICATION EXAMPLE

In order to present a constructive, practical application of the 1T-2F logic consider monitoring and diagnosis of the classical, non-trivial benchmark system being a multiplier-adder network presented in [22] and further frequently explored in the domain literature, e.g. [9, 2, 1]. The system schema is presented in Figure 1. The system

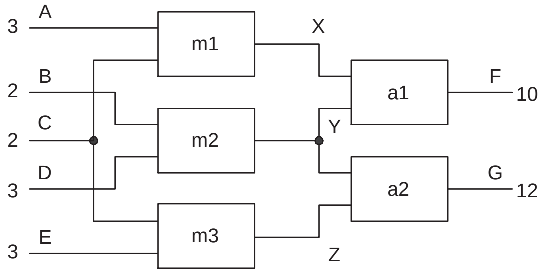

Figure 1. An example arithmetic system

is composed of two layers. The first one contains three multipliers $m1$, $m2$, and $m3$, and receives five input signals A, B, C, D and E. The second layer is composed of two adders, namely $a1$ and $a2$, and produces two output values of F and G. Only inputs (of the first layer) and outputs (of the second layer) are *directly observable*. The intermediate variables X, Y, and Z are hidden; one cannot measure them.

The observed state of the system is as follows: A=3, B=2, C=2, D=3 and E=3. If the system works correctly — the outputs should be F=12 and G=12 (with X=6, Y=6 and Z=6). Since the current value of F is incorrect, namely F=10, the system is faulty. In fact, *at least one* of its components must be faulty. Note that the fault may consist in *lowering* or *increasing* the signal level.

In this classical case, i.e. F being faulty and G correct, the final minimal diagnoses are calculated as minimal hitting sets of the following conflict sets: $C_1 = \{m1, m2, a1\}$ and $C_2 = \{m1, m3, a1, a2\}$ [22]. There are four potential diagnoses: $D_1 = \{m1\}$, $D_2 = \{a1\}$, $D_3 = \{a2, m2\}$ and $D_4 = \{m2, m3\}$; note that the last two, two-component diagnoses take into account the phenomenon of *compensation* in order to recover the correct value of G.

Now, one can ask for a *more precise definition of faults*. In fact, diagnosis $D = \{p\}$ can be considered as a statement that p is false; in the presented 1T-2F logic it can be $p[-]$ or $p[+]$, since $\rightharpoondown p[0] = p[-]$ and $\rightharpoonup p[0] = p[+]$; the obvious meaning is that the incorrect value of the signal is *lower* or *higher* then the correct one. A statement such as $p[-]$ or $p[+]$ constitute in fact a *qualitative diagnosis* [19] which provides reacher information about the nature of the fault. Such more precise information can be used to eliminate inadmissible diagnoses (e.g. a fault battery always produces lower voltage). In the example, the four classical diagnoses lead to as many as 12 potential qualitative diagnoses: $\{m1[-]\}$, $\{m1[+]\}$, $\{a1[-]\}$,

$\{a1[+]\}$, $\{a2[-], m2[-]\}$, $\{a2[-], m2[+]\}$, $\{a2[+], m2[-]\}$, $\{a2[+], m2[+]\}$, $\{m2[-], m3[-]\}$, $\{m2[-], m3[+]\}$, $\{m2[+], m3[-]\}$, $\{m2[+], m3[+]\}$.

More formally, if $p[0]$ denotes correct work of a component, $\neg p[0] = p[-, +]$ denotes possible faults; since p can take a unique truth value at a time, two candidate diagnoses, $p[-]$ and $p[+]$ should be considered. In case of multiple diagnosis $\{p_1, p_2, \ldots, p_k\}$ we have the Cartesian product $\mathsf{X}_{i=1..k}\{p_i[-], p_i[+]\}$.

Now, consider a single component p, with two inputs x_1 and x_2, and single output y — such as a multiplier or adder in the example system. The logical model of it producing (hopefully correct) output can be expressed in the form: $x_1 \wedge x_2 \wedge p \models y$ with the interpretation, that if the inputs are correct, and the component works correctly, the output should be correct as well.

Now, consider a single potential qualitative diagnosis $\{m1[-]\}$. Combined with inputs $A[0]$ and $B[0]$ we conclude that $X[-]$. This, combined with $Y[0]$ and $a1[0]$ gives $F[-]$. Hence diagnosis $\{m1[-]\}$ is confirmed; it is also consistent with other observations, since $Y[0]$, $Z[0]$ and $a2[0]$ gives $G[0]$ consistent with the expected value of G.

On th other hand, potential diagnosis $\{m1[+]\}$ with $A[0]$ and $B[0]$ leads to $X[+]$, and in consequence, having $Y[0]$ and $a1[0]$ we conclude $F[+]$. Hence, diagnosis $\{m1[+]\}$ is rejected; it is inconsistent with observations.

In an analogous way, $a1[-]$ is confirmed, while $a1[+]$ must be rejected.

Now, consider potential diagnosis with two components: $\{a2[-], m2[-]\}$. $m2[-]$ combined with $B[0]$ and $D[0]$ leads to $Y[-]$. This combined with $Z[0]$ and $a2[-]$ leads to $G[-]$. Hence diagnosis $\{a2[-], m2[-]\}$ must be rejected; it is inconsistent with observations. On the other hand, $\{a2[+], m2[-]\}$ leads to $Y[-]$ (as before), but $Y[-]$ combined with $Z[0]$ and $a2[+]$ leads to $G[?]$ — no inconsistency is detected on G. Further, $Y[-]$ combined with $X[0]$ and $a1[0]$ gives $F[-]$ which explains the observed fault.

In a similar way diagnosis $\{m2[-], m3[+]\}$ is proved admissible, while the other three are rejected as inconsistent.

Finally, we arrive at the following set of admissible qualitative diagnoses: $\{m1[-]\}$, $a1[-]$, $\{a2[+], m2[-]\}$, $\{m2[-], m3[+]\}$. Note that, we have obtained more precise diagnoses at no cost; just a more precise logic for modeling the nature of faults has been employed. Note also, that if auxiliary knowledge about possible fault types is available, further refinement through inconsistency elimination can be pursued.

6 CONCLUSIONS

An idea of a new, inspiring 1T-2F logic has been put forward. It is aimed at more precise modeling of systems for tasks such as monitoring, control or diagnosis. New interpretation of negation have been introduced and application example has been shown.

Note that this kind of negation can be also considered as *constructive negation*; instead of saying 'anything, but not p' (without bothering, if and what can this mean), we insist on pointing to a different, but specific value.

The 1T-2F logic seems to have potential for development of further kT-mM-nF type logics with k true values, m middle values and n different false values; the k true can be interpreted as different OK working modes, while the m middle vales — as some partial faults.

REFERENCES

[1] Marie-Odile Cordier et al., 'Ai and automatic control approaches of model-based diagnosis: Links and underlying hypotheses', in *SAFE-PROCESS'2000*, pp. 274–279. IFAC, (2000).

[2] Marie-Odile Cordier et al., 'A comparative analysis of ai and control theory approaches to model-based diagnosis', in *ECAI'2000. 14th European Conference on Artificial Intelligence*, ed., Werner Horn, pp. 136–140. IOS Press, (2000).

[3] R. Davis and W. Hamscher, 'Model-based reasoning: Troubleshooting', in *Readings in Model-Based Diagnosis*, eds., W. Hamscher, L. Console, and J. de Kleer, 3–24, Morgan Kaufmann Publishers, San Mateo, CA, (1992).

[4] P.M. Frank, 'Fault diagnosis in dynamic systems using analitical and knowledge-based redundancy–a survey and some new results', *Automatica*, **26**(3), 459–474, (1990).

[5] P.M. Frank, 'Analytical and qualitative model-based fault diagnosis - a survey and some new results', *European Journal of Control*, **2**, 6–28, (1996).

[6] P. Fuster-Parra, *A Model for Causal Diagnostic Reasoning. Extended Inference Modes and Efficiency Problems. – Ph.D. Thesis*, University of Balearic Islands, Palma de Mallorca, Spain, 1996.

[7] B. Górny and A. Ligęza, 'Model-based diagnosis of dynamic systems: Systematic conflict generation', in *Paper presented at the Conference on Model-Based Reasoning, Scientific Discovery, Technological Innovations, Values, MBR'2001*, Kluwer Academic Publishers, Pavia, Italy, (May 2001).

[8] Bartłomiej Górny, *Consistency-Based Reasoning in Model-Based Diagnosis*, Ph.D. dissertation, AGH, Kraków, Poland, 2001.

[9] *Readings in Model-Based Diagnosis*, eds., W. Hamscher, L. Console, and J. de Kleer, Morgan Kaufmann, San Mateo, CA, 1992.

[10] *Fault Diagnosis. Models, Artificial Intelligence, Applications*, eds., J. Korbicz, J.M. Kościelny, Z. Kowalczuk, and W. Cholewa, Springer-Verlag, Berlin, 2004.

[11] Jan Maciej Kościelny, *Diagnostyka zautomatyzowanych procesów przemysłowych*, Problemy Współczesnej Nauki. Teoria i Zastosowania. Automatyka, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2001.

[12] Jan Maciej Kościelny, *Methodology of Process Diagnosis*, chapter 3, 57–114, In: [10], Springer-Verlag, 2004.

[13] Jan Maciej Kościelny, *Models in Process Diagnosis*, chapter 2, 29–43, In: [10], Springer-Verlag, 2004.

[14] A. Ligęza, 'A note on systematic conflict generation in ca-en-type causal structures', *LAAS Report No. 96317*, (1996).

[15] A. Ligęza and P. Fuster-Parra, 'And/or/not causal graphs – a model for diagnostic reasoning', *Applied Mathematics and Computer Science*, **Vol. 7, No. 1**, 185–203, (1997).

[16] A. Ligęza and B. Górny, 'Systematic conflict generation in model-based diagnosis', in *Safeprocess'2000. 4th IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes*, ed., A. M. Edelmayer, volume 2, 1103–1108, IFAC, Budapest, (2000).

[17] Antoni Ligęza, *Selected Methods of Knowledge Engineering in System Diagnosis*, chapter 16, 633–668, In: [10], Springer-Verlag, 2004.

[18] Antoni Ligęza, 'A constraint satisfaction framework for diagnostic problems', in *Diagnosis of Processes and Systems*, ed., Zdzisļaw Kowalczuk, Control and Computer Science. Information Technology, Control Theory, Fault and System Diagnosis, pp. 255–262, Gdańsk, (2009). Pomeranian Science and Technology Publisher PWNT.

[19] Antoni Ligęza and Jan M. Kościelny, 'A new approach to multiple fault diagnosis. combination of diagnostic matrices, graphs, algebraic and rule-based models. the case of two-layer models', *Int. J. Appl. Math. Comput. Sci.*, **18**(4), 465–476, (2008).

[20] J. A. Reggia, D. S. Nau, and P. Y. Wang, 'Diagnostic expert system based on a set covering model', *International Journal on Man-Machine Studies*, **19**, 437–460, (1983).

[21] J. A. Reggia, D. S. Nau, and P. Y. Wang, 'A formal model of diagnostic inference. problem formulation and decomposition', *Information Sciences*, **37**, 227–256, (1985).

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

[23] *Knowledge-Based System Diagnosis, Supervision and Control*, ed., S.G. Tzafestas, Plenum Press, New York, London, 1989.

[24] S.G. Tzafestas, 'System fault diagnosis using the knowledge-based methodology', in *A chapter in [23]*, 509–595, Plenum Press, (1989).

Modeling and diagnosis of dynamic systems from timed observations

Ismail Fakhfakh [1] and **Marc Le Goc** [2] and **Lucile Torres** [2] and **Corinne Curt** [1]

Abstract. This paper proposes the use of the Timed Observation theory as a powerful framework for model-based diagnosis. In fact, this theory provides a global formalism for modeling a dynamic system (TOM4D), for characterizing and computing diagnoses of the system under investigation

1 INTRODUCTION

In the last two decades model-based diagnosis has been an important research area where numerous new methodologies and formalisms have been proposed, studied and experimented ([3] and [9]). This is motivated by the practical need for ensuring the correct and safe functioning of large complex systems. These frameworks have been created (i) to provide semantics for the diagnosis problem solving, (ii) to analyze the properties and to characterize the diagnosis reasoning and (iii) to give modeling principles.

In dynamic systems, the observation is timed unlike in static systems where the observations are given at only one point of time. This is restrictive in several fields. The extension of the problem poses many problems with the existing approaches. Since (Reiter, 1987), most of the frameworks are based on the logic formalism. Despite of the important contributions in the domain of temporal logics, there is still a difficulty to take into account the time of the observations in the diagnosis reasoning. Later, the Discret Event System formalism has been used to diagnose dynamic systems [1]. One basic difficulty that arises is then the definition of the observations. Cordier [4] proposes to slice off the flow of the measurements into temporal windows to define the observations within these slices and to compute the diagnosis incrementally using the observations of the successive slices. This approach is applied only to D.E.S and is seldom used in real cases. One of the problems with this kind of approach is to define the size of the slices so that the relevant observations can be perceived: there is no a priori reason for the observations to be synchronized with the slicing algorithm. In other words, the slicing algorithm can mask pertinent observations and, within a slice, the observations must be ordered to be taken into account in a model. These difficulties are classical with discrete time systems. To avoid these problems, Le Goc [8] proposes to define observations time-stamped with clocks in time continuous. The Timed Observation Theory of Le Goc [8] provides a general mathematical framework for modeling dynamic processes from timed data. The application of this framework to diagnosis has given birth to a modeling methodology for diagnosis TOM4D (Timed Observation Theory for Diagnosis). The aim of the modeling methodology is to provide an efficient diagnosis based on models built at the same abstraction level as these of the experts.

In this paper, after a brief presentation of the Timed Observation Theory and the TOM4D method (Sections 2 and 3), we show how TOM4D supports the modeling of complex physical systems. In sections 4 and 5, we show how the models can be used to characterize the diagnosis and we demonstrate that the diagnosis can be computed easier using the TOM4D models (section 6). We apply the modeling approach and the diagnosis algorithm to an hydraulic system. Finally, Section 7 provides conclusions and proposes some perspectives to this work.

2 THEORY OF TIMED OBSERVATIONS

Le Goc's Timed Observation Theory extends Shannon's Theory of Communication to timed data and offers a unique frame for Markov Chains and Poisson Theories. It also extends the Logical Theory of Diagnosis to timed observations. This theory considers that the timed messages of a serie are written in a database by a program called a Monitoring Cognitive agent (MCA), which monitors a dynamic sytem. A dynamic system is a process $Pr(t)$=$\{x_1(t)$, $x_2(t)$, ..., $x_n(t)\}$ defined as an arbitrary set made of time functions $x_i(t)$ defined on the real set denoted $\Re$ (i.e. $\forall$ t $\in \Re$, $x_i(t) \in \Re$).

This theory defines a timed observation in the following way [8]. Given a set $Pr(t)$=$\{x_1(t)$, $x_2(t)$, ..., $x_n(t)\}$ of time functions the evolution of which are observed by a program Θ; let $X = \{x_1$, x_2, ..., $x_n\}$ be the corresponding set of variable names; let $\Delta = \bigcup_{\forall x_i \in X} \Delta_{x_i}$ each Δ_{x_i} =$\{\delta_1^i$, δ_2^i, ..., $\delta_m^i\}$ being a set of constants denoting the possible values for x_i; let Γ=$\{t_k\}_{t_k \in \Re}$ be a set of arbitrary time instants.

Definition 1 (A Timed Observation). *A timed observation $o(t_k) \equiv (\delta_j^i, t_k)$, made by a program θ when observing a time function $x_i(t)$ at time $t_k \in \Gamma$, is the assignation of the values v=x_i, δv=δ_j^i and t=t_k to a predicate $\Theta(v, \delta v, t)$ so that: $\Theta(x_i, \delta_j^i, t_k)$.*

Conceptually, the θ program applies the spatial segmentation principle: a value δ_j^i is assigned to a variable x_i whenever the value of its corresponding time function $x_i(t)$ enters in a range $[\psi^i, \psi^{i+1}[$, where ψ^i is a threshold for $x_i(t)$ (i.e. $\psi^i \in \Re$). This means that the values are assigned to the variables with a program (or a human) the basic specification of which is the following (cf. [7] for examples of more complex spatial segmentation algorithms):

$\forall$ k $\in \mathbb{N}$, $x_i(t_k) \geq \psi^i \wedge x_i(t_{k-1}) < \psi^i \Rightarrow o(t_k) \equiv (\delta_j^i, t_k) \wedge t_k \in \Gamma$

In practice, each time t_k the predicate $\Theta(x_i, \delta_j^i, t_k)$ is assigned, the program θ (or a human) writes a couple (δ_j^i, t_k) in a database, a datalog or a simple document. As a consequence, to any timed observation o(t_k)$\equiv$ (δ_j^i, t_k) corresponds an assigned predicate $\Theta(x_i$, δ_j^i,

[1] IRSTEA, 3275 route de Cézanne - CS 40061, Aix-en-Provence, France
[2] Aix-Marseille Université, LSIS, 13397 Marseille, France

t_k). [8] shows that this predicate can always be interpreted as the "Equal" predicate so that: $\Theta(x_i, \delta_j^i, t_k) \equiv$ Equal$(x_i, \delta_j^i, t_k) \Leftrightarrow x_i(t_k) \in [\psi^i, \psi^{i+1}[$. Such an assigned predicate is often represented in the expert's language under the form of the assignation of the value δ_j^i to the variable x_i at t_k: $x_i(t_k)$=δ_j^i. The value δ_j^i can therefore then be considered as a symbol denoting the range $[\psi^i, \psi^{i+1}[$. This leads to define the notion of class of observations.

Definition 2. *An observation class C^i=$\{(x_i, \delta_j^i), (x_{i+1}, \delta_{j+1}^{i+1}), ..., (x_{i+n}, \delta_{j+n}^{i+n}\}$ is a set of couples (x_i, δ_j^i) associating a variable x_i, eventually unknown, with a constant δ_{j+k}^{i+k}.*

In other words, an observation class C_i associates variables $x_i \in X$ with constants $\delta_j^i \in \Delta_{x_i}$. This leads to the following property:

Proposition 2.1. *Each timed observation $o(t_k) \equiv (\delta_j^i, t_k)$ corresponds to an occurrence of an observation class $C_i = \{(x_i, \delta_j^i)\}$.*

In practical applications, the observation classes are usually defined as a singleton of the form $C_i = \{(x_i, \delta_j^i)\}$. These definition allow defining a modeling methodology for diagnosis.

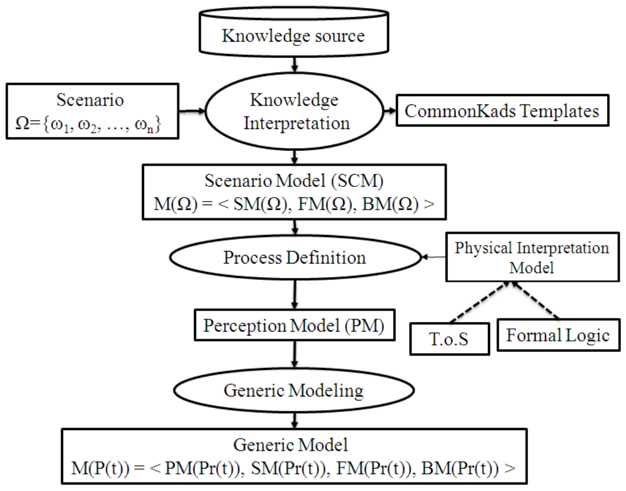

Figure 1. TOM4D Modeling Process

3 MODELING APPROACH FOR DIAGNOSIS : TOM4D

TOM4D is a modeling methodology for dynamic systems focused on timed observations. The objective of this methodology is to produce a suitable model for dynamic process diagnosis from timed observations and experts' a priori knowledge. TOM4D relies on the idea that experts use an implicit model to both formulate the knowledge about the process and diagnose it. It is a multi-model approach that combines CommonKads templates [11] with the conceptual framework proposed in [12] and the tetrahedron of states (T.o.S), [10], [2]. These elements are merged according to the Timed Observations Theory [8].

The TOM4D methodology is based on the notion of observation class $C_i = \{(x_i, \delta_j^i)\}$ and associates the variable x_i of each observation class C_i with one and only one component c_i. This means that the values δ_j^i a variable x_i can take over time is the result of a couple ($\theta(\Delta_{x_i})$, x_i(t)) made with a program $\theta(\Delta_{x_i})$ that observes the evolutions of a time function $x_i(t)$ and write a timed observation o(t_k) $\equiv (\delta_j^i, t_k)$ whenever a predicate $\Theta(x_i, \delta_j^i, t_k)$ is assigned. In other words, $x_i(t)$ is the signal provided by some sensors associated with a component c_i. This allows to organize the available knowledge about a process $Pr(t)$ according to (i) a Perception Model $PM(Pr(t))$ defining the process as an arbitrary set made of time functions $x{-}i(t)$ and its operating goals and its normal and abnormal behaviors, (ii) a Structural Model $SM(Pr(t))$ defining the components of the process and their relations, (iii) a Functional Model $FM(Pr(t))$ defining the relations between the values of the process variables (i.e. their definition domain) with a set of mathematical functions, and (iv) a Behavior Model $BM(Pr(t))$ defining the timed observation classes firing the evolutions of the time functions of $Pr(t)$.

Figure 1 describes the three main steps of the TOM4D modeling process: Knowledge Interpretation, Process Definition and Generic Modeling. The aim of this process is to produce a coherent generic model $M(Pr(t))$ = $<$ $PM(Pr(t))$, $SM(Pr(t))$, $FM(Pr(t))$, $BM(Pr(t))$$>$ from the available knowledge and data.

The Knowledge Interpretation step uses a CommonKADS template to interpret and to organize the available knowledge about a dynamic system. This knowledge is provided by a knowledge source (an expert, a set of documents, etc) and when possible, at least one scenario. This first step aims at producing a scenario model $M(\Omega) =< SM(\Omega), FM(\Omega), BM(\Omega) >$ of the system that is coherent with the available knowledge about its evolution over time. This model is used in the Process Definition step to provide a definition of the process under the form of a perception model $PM(Pr(t))$. This is made with the use of the tetrahedron of states to provide a physical dimension to each variable of the process and with the use of formal logic to define its operating goals and its normal and abnormal behaviors. The aim of this step is to control the way the semantics of the available knowledge is introduced in the model to avoid the potential representation errors. The Perception Model $PM(t)$ defined, the Generic Modeling step aims at defining an abstract representation of the dynamic system where the different terms of the available knowledge are reified through a set of relations. This paper being focused on the use of the resulting model $BM(Pr(t))$, the interested reader is invited to see [9], [5] or [6] for further details about TOM4D.

A TOM4D behavior model $BM(Pr(t))$ describes the possible sequences of observation classes that can occur and therefore the discernible states between them.

Definition 3. *A behavior model $BM(Pr(t))$ of a dynamic process $Pr(t)$ is a 3-tuple $< S, C, \gamma >$ where:*

- *$S = \{s : X \to \Delta | s(x_i) = \delta, x_i \in X, \delta \in \Delta\}$ is a set of functions which characterize the discernible states of the process $Pr(t)$,*
- *C is a set of observation classes, where an observation class associated with a variable $x_i \in X$ is a set $C^i = \{(x_i, \delta) | \delta \in \Delta^{x_i}\}$ containing only one element (i.e. a singleton),*
- *$\gamma : S \times C \to S$ is a function of discernible state transition.*

Given a sequence $\omega = \{o(k)\}$ of observation class occurrence $o(k) \equiv (\delta^i, t_k)$, a transition from a discernible state s_i to the discernible state s_j is triggered when:

- there is an occurrence $o(k) \equiv (\delta^i, t_k)$ of class C^i in ω;
- the current state $s(t)$ of the finite state machine implementing $BM(Pr(t))$ is the discernible state s_i (i.e. $s(t) = s_i$);
- there exists an assignment $\gamma(C^k, s_i) = s_j$.

The observation classes being singletons, an occurrence of an observation class (i.e. a timed observation $o(k) \equiv (\delta^y, t_k)$) corresponds to the assignation of a particular value δ^y to a variable x_i of $Pr(t)$.

4 SEQUENTIAL BINARY RELATIONS

The important point is that a state transition in a finite state machine implementing a TOM4D behavior model $BM(Pr(t))$ can occur if and only if there exist two assignations $s_i = \gamma(C^x, s_{k-1})$ and $s_k = \gamma(C^y, s_i)$ in $BM(Pr(t))$.

Definition 4. *Given a TOM4D behavior model $BM(Pr(t)) =< S, C, \gamma >$, a sequential binary relation $r(C^x, C^y, s_i)$ between two observations classes C^x and C^y, labelled with a discernible state s_i, exists iff: $\exists s_{k-1}, s_i, s_k \in S, s_i = \gamma(C^x, s_{k-1}) \wedge s_k = \gamma(C^y, s_i)$.*

A sequential binary relation between two observation classes $r(C^x, C^y, s_i)$ is an oriented (sequential) relation between two observation classes $C^x = \{(x, \delta^x)\}$ and $C^y = \{(y, \delta^y)\}$ that is linked with a discernible state s_i. This latter can correspond to the current state of a finite state machine implementing a TOM4D behavior model $BM(Pr(t))$ after observing an occurrence $C^x(t_k) = (\delta^x, t_k)$ of the "input" observation class C^x and before observing the occurrence $C^y(t_{k+1}) = (\delta^y, t_{k+1})$ of the "output" observation class C^y.

The γ function defines then the possible sequential relations between two observation classes:

Proposition 4.1. *Two assignments $s_i = \gamma(C^x, s_{k-1})$ and $s_k = \gamma(C^y, s_i)$ define a sequential binary relation $r(C^x, C^y, s_j)$ between two classes C^x and C^y labelled with a discernible state s_i.*

In other words, a TOM4D behavior model $BM(Pr(t)) =< S, C, \gamma >$ specifies a graph between the set C of observation classes. This graph is used to control the diagnosis reasoning.

A class graph C-Graph is a set $G_C = \{..., r_i(C^x, C^y, s_{x,y}), ...\}$, $i = 1...n$, of sequential binary relations of the form $r(C^i, C^o, s_{io})$ between an input observation class C^i and an output observation class C^o labelled with a discernible state s_{i_o}. The C-Graph is built from a TOM4D generic behavior model generated with the following algorithm.

```
Algorithm: Generate-C-Graph $G_C = \{r_i\}$
input: a behavior Model $BM(Pr(t)) =< S, C, \gamma >$
output: a C-Graph $G_C = \{r_i\}, r_i \equiv r(C^x, C^y, s_i)$
1.$G_C = \Phi$
2.$\forall s_i \in S$
2.1.$\exists s_n, s_m \in S$,
$s_n = \gamma(s_i, C_x) = s_n \wedge s_m = \gamma(s_n, C^y)$
$\Rightarrow G_C = G_C \cup r(C^x, C^y, s_i)$;
3.$Return G_C$
```

The C-Graph G_C describes the complete process behavior in terms of observation class. This means that a path in this graph describes a particular behavior of the process. Such a path correspond to a suite of discernable states in the behavior model $BM(Pr(t))$. So looking for a particular suite of discernable states in $BM(Pr(t))$ corresponds to look for a particular path in the associated C-Graph G_C:

Definition 5. *A class path P^C is a sub-graph of a C-Graph G_C made with a suite $P^C = (r_{i,i+1})$, $i = 1...n$ of n sequential binarys relation $r_{i,i+1}$ of the form $r(C^i, C^{i+1}, s_{i,i+1})$.*

In other words, the general form of a class path P^C is the following: ($r_1(C^{i_1}, C^{i_2}, s_{i_1,i_2})$, $r_2(C^{i_2}, C^{i_3}, s_{i_2,i_3})$, ..., $r_n(C^{i_n}, C^{i_{n+1}}, s_{i_n,i_{n+1}})$).

Because the timed observations provided by a MCA $\Theta(X, \Delta)$ are the occurrences of the observation classes of the set C of a TOM4D behavior model $BM(Pr(t))$, it is simpler to look for a class path in the C-Graph and then to look for the corresponding state path, rather that trying to directly build the suite of states from the suite of observations. This idea is the basis of the proposed diagnosis algorithm.

5 DIAGNOSING WITH C-GRAPHS

According to The timed Observation Theory [8], the timed observations are provided by a MCA $\theta(X, \Delta)$ that assumes the online supervision of a dynamic process $Pr(t)$. Diagnosis is performed starting from a sequence $\omega = \{o(t_k)\}$ of timed observations and a TOM4D process model $M(Pr(t))$. It consists in explaining the timed observations of ω written by MCA $\Theta(X, \Delta)$ during a period $[t_0, t_n]$.

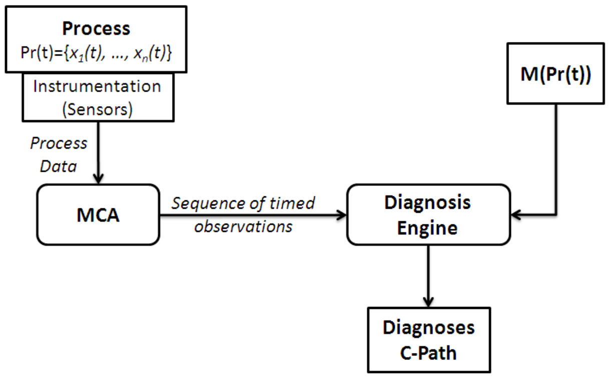
Process
Pr(t)={x1(t), ..., xn(t)}
Instrumentation (Sensors)
Process Data
MCA
Sequence of timed observations
Diagnosis Engine
M(Pr(t))
Diagnoses C-Path

Figure 2. Diagnosis Engine

Consequently, the diagnosis aims at generating the minimal set D of class paths P^C that are compatible with the timed observations of ω (cf. Figure 2) and the C-Graph derived from the behavior model $BM(Pr(t))$ of the TOM4D process model $M(Pr(t))$.

Definition 6 (Diagnosis Definition). *Given a C-Graph $G_C = \{..., r(C^x, C^y, s_i), ...\}$ and a suite $\omega = \{o(t_0), ..., o(t_n)\}$ of $n + 1$ timed observations recorded during the period $[t_0, t_n]$, a diagnosis at time $t \in [t_0, t_n]$ is the minimal set $D(t) = \{P^C\}$ of class paths P^C that are consistent with G_C and ω.*

$$(\omega, G_C) \rightarrow D(t_n) \tag{1}$$

The algorithm of computing the minimal set D of class path P^C from a C-graph G_C and a sequence ω of timed observations is made with a loop on each timed observation $o(k) \in \omega$ and acts with three main steps: (i) remove the paths of D that are no more coherent with $o(k)$, (ii) extend each path in the resulting set D with the right sequential relations from G_C and (iii) initialize the set D when it is empty (at the first loop or if there are no more paths that are coherent with ω. The algorithm also uses three functions: "$obsClassOf(o)$" to get the class of a timed observation, "$rightestRelationOf(P)$" to get the right most sequential binary relations of a class path and "$rightRelations(r(C^i, C^o, s_{io}), G_C)$" to get the set of sequential binary relation corresponding to the successor of a particular sequential binary relation $r(C^i, C^o, s_{io})$ in a C-Graph G_C.

Algorithm: Generate-Class-Path
input : a C-Graph G_C and a sequence $\omega = \{o(t_k)\}$ of n timed observations
Output : a set D of class paths consistent G_C and ω

```
1.$D \leftarrow \{\phi\}$
//Loop on the timed observations of $\omega$
//$\Gamma(\omega)$ is the set of time-stamp of the timed observations of $\omega$
2.$\forall t_k \in \Gamma(\omega)$
//Compute the set $C$ of the classes occured at $t_k$
2.1.$\forall o(t_k) \equiv (\delta^i, t_k) \in \omega, ObsClassOf(o(t_k)) \in C$
//Compute $D$ for the set $C$ at time $t_k$
2.2.$D = computeD(D, G_C, C)$
3.$ReturnD$
```

Algorithm: computeD
input : a set D of C-Path, a C-Graph G_C and a set C of observation classes
Output : the upated set D

```
1.$d \leftarrow \{\phi\}$
//Loop on the observation classes of $C$
2.$\forall c \in C$
2.1.$d \leftarrow d \cup computeCPaths(D, G_C, c)$
3.$D \leftarrow d$
//If $D$ is empty, initialise $D$ with $C$ and $G_C$
4.$D = \{\phi\} \Rightarrow D \leftarrow initCPath(G_C, C)$
5.$ReturnD$
```

Algorithm: computeCPaths
input : a set D of C-Paths, a C-Graph G_C and an observation class c
Output : the updated set D

```
1.$D_1 = \{\phi\}$ //Working set of C-Path
//Remove from $D$ the paths that are not compatible with $c$
2.$\forall P \in D$
2.2.$r(C^i, C^o, s_{io}) \leftarrow rightestRelationOf(P)$
2.3.$C^o = c \Rightarrow D_1 = D_1 + P$
3.$D \leftarrow D_1$ //$D$ contains the C-Paths compatible with c
//Extends each path of $D$ with the right sequential relations
4.$D_1 = \{\phi\}$ //Reset the working set $D_1$
4.$\forall P \in D$
4.1.$r(C^i, C^o, s_{io}) \leftarrow rightestRelationOf(P)$
//Get the relations from $G_C$
4.2.$R = rightRelations(r(C^i, C^o, s_{io}), G_C)$
4.3.$\forall r \in R$
4.3.1.$P_1 = P + r$ //Create a new extended path for $P$
4.3.2.$D_1 = D_1 + P_1$ //Add the new path in $D_1$
5.$ReturnD_1$
```

Algorithm: initCPath
input : a C-Graph G_C and a set C of observation classes
Output : a set D of sequential binary relations consistent with C

```
1.$\forall c \in C$
1.1$\forall r(C^i, C^o, s_{io}) \in G_C$,
1.2$C^o = c \Rightarrow \{r(C^i, C^o, s_{io})\} \in D$
2.$ReturnD$
```

The next section illustrates this algorithm on the (simple) device of Figure 3 studied in [3]. It is to note that this algorithm can easily be extended to simultaneous timed observations that can occur in large and complex systems. In other hand, the lack of timed observations leads the algorithm to remove the C-Paths that are no more consistent with the suite of timed observations. It can also be extended to use the functional model $FM(Pr(t))$ to distinguish between a true lack of timed observation and an inconsistency between the sequence of timed observations and the behavior model $BM(Pr(t))$.

6 APPLICATION

[3] describes the example with the following terms: the system is formed by a pump P which delivers water to a tank TA via a pipe PI; another tank CO is used as a collector for water that may leak from the pipe. The pump is always on and supplied of water. The pipe PI can be ok (delivering to the tank the water it receives from the pump) or leaking (in this case we assume that it delivers to the tank a low output when receiving a normal or low input, and no output when receiving no input). The tanks TA and CO are simply receive water. We assume that three sensors are available (see the eyes in Figure 3): $flow_p$ measures the flow from the pump, which can be normal (nrm_p), low (low_p), or zero (zro_p); $level_{TA}$ measures the level of the water in TA, which can be normal (nrm_{ta}), low (low_{ta}), or zero (zro_{ta}); $level_{CO}$ records the presence of water in CO, either present (pre_{co}) or absent (abs_{co}).

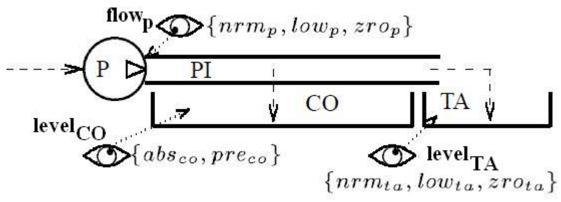

Figure 3. Hydraulic system

According to the TOM4D methodology, the system is a hydraulic process $Pr(t) = \{x_1(t), x_2(t), x_3(t)\}$ made with three variables (cf. the hydraulic T.o.S): $x_1(t)$ is a volume variable, $x_2(t)$ and $x_3(t)$ are two outflow variables. The analysis of the system description shows that $x_2(t)$ represents a normal outflow and $x_3(t)$ represents an abnormal outflow corresponding to water leakage. Table 1 shows the variable-value association and the physical interpretation of the variables. The corresponding set of observation classes is given in Table 2 and the discernible states are provided in Table 3. The reader interested with the application of the TOM4D methodology on this example is invited to refer to [5].

Variables x	Physical interpretation	value interpretation	Abstract value δ_i
x_1	$Volume$	$normal$,	2,
		low,	1,
		$zero$	0
x_2	$normal$ $outflow$	$normal$,	2,
		low,	1,
		$zero$	0
x_3	$abnormal$ $outflow$	$presence$,	2,
		$absence$	1

Table 1. Variable-Value Association for the Hydraulic System

$C_1^1 = \{(x_1,0)\}$	$C_2^1 = \{(x_1,1)\}$	$C_3^1 = \{(x_1, 2)\}$
$C_1^2 = \{(x_2,0)\}$	$C_2^2 = \{(x_2,1)\}$	$C_3^2 = \{(x_2,2)\}$
$C_1^3 = \{(x_3,1)\}$	$C_2^3 = \{(x_3,2)\}$	

Table 2. Timed Observation Classes

States	x_1	x_2	x_3	*States*	x_1	x_2	x_3
s_0	0	0	1	s_1	1	0	1
s_2	2	0	1	s_4	1	1	1
s_5	2	1	1	s_8	2	2	1
s_9	0	0	2	s_{10}	1	0	2
s_{11}	2	0	2	s_{13}	1	1	2
s_{14}	2	1	2	s_{17}	2	2	2

Table 3. The set of discernible states for the Hydraulic System

Figure 4 shows a graphical representation of the behavior model $BM(Pr(t))$ of the hydraulic system. The "$Generate-C-Graph$" algorithm of section 4 produce the C-Graph G_C of Figure 5.

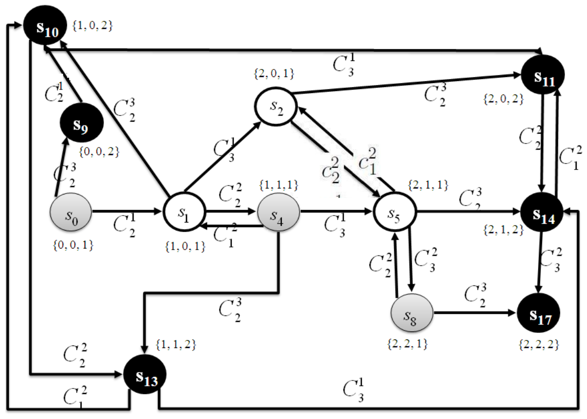

Figure 4. behavior Model of the hydraulic system

To illustrate the "$Generate - Class - Path$" algorithm of the previous section, let us consider the following sequence of timed observations: ω = { $o_{x_2}(t_0) \equiv$ (1, t_0), $o_{x_3}(t_1) \equiv$ (2, t_1), $o_{x_2}(t_2) \equiv$ (0, t_2) , $o_{x_1}(t_3) \equiv$ (2, t_3)} (We consider that $t_i \leq t_{i+1}$). According to the table 2, the observation class associated to the first timed observation $o_{x_2}(t_0)$ of ω is C_2^2. The set D being empty, the two first steps of the algorithm do nothing but the third step initializes D with the Algorithm *"initCPath"* that is to say finds the set of binary relation that are of the form r(C^0, C_2^2, s_{i_0}) so that D = { {r(C_2^1 ,C_2^2, s_1)}, {r(C_1^2 ,C_2^2, s_1)}, {r(C_1^2 ,C_2^2, s_2)}, {r(C_3^1 ,C_2^2, s_2)}, {r(C_3^2 ,C_2^2, s_8)}, {r(C_1^2 ,C_2^2, s_{11})}, {r(C_2^3 ,C_2^2, s_{11})}, {r(C_3^1 ,C_2^2, s_{11})}, {r(C_1^2 ,C_2^2, s_{10})}, {r(C_2^3 ,C_2^2, s_{10})}, {r(C_2^1 ,C_2^2, s_{10})}}.

The observation class of the second timed observation $o_{x_3}(t_1) \equiv$ (2, t_1), $o_{x_2}(t_2)$ being C_2^3, the next step of the algorithm removes the paths of D that are no more coherent with $o_{x_3}(t_1)$ and extends the rest of paths with the right sequential relations from G_C (cf. Figure

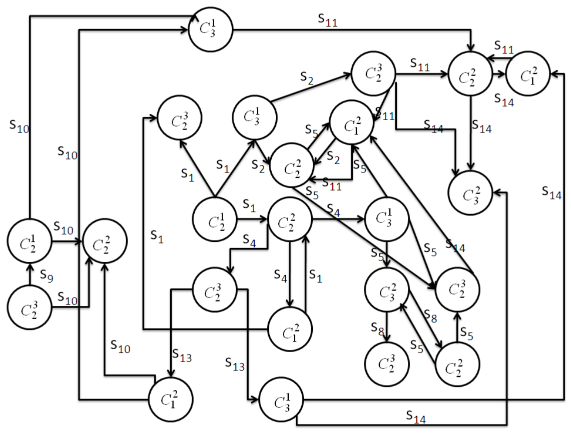

Figure 5. C-Graph of the hydraulic system

5) so that $D = \{ \{r(C_2^1, C_2^2, s_1), r(C_2^2, C_2^3, s_4)\}, \{r(C_1^2, C_2^2, s_1), r(C_2^2, C_2^3, s_4)\}, \{r(C_1^2, C_2^2, s_{11}), r(C_2^2, C_2^3, s_{14})\}, \{r(C_1^2, C_2^2, s_2), r(C_2^2, C_2^3, s_5)\}, \{r(C_3^1, C_2^2, s_2), r(C_2^2, C_2^3, s_8)\}, \{r(C_3^2, C_2^2, s_5), r(C_2^2, C_2^3, s_5)\} \}$. Doing so, the algorithm finds only two C-Paths that are consistent with all the timed observations of ω (cf. Fig 6). The dark circle means that the new observation class is inconsistent with the defined C-Path (there is no relation between the last observation class and the new observation class).

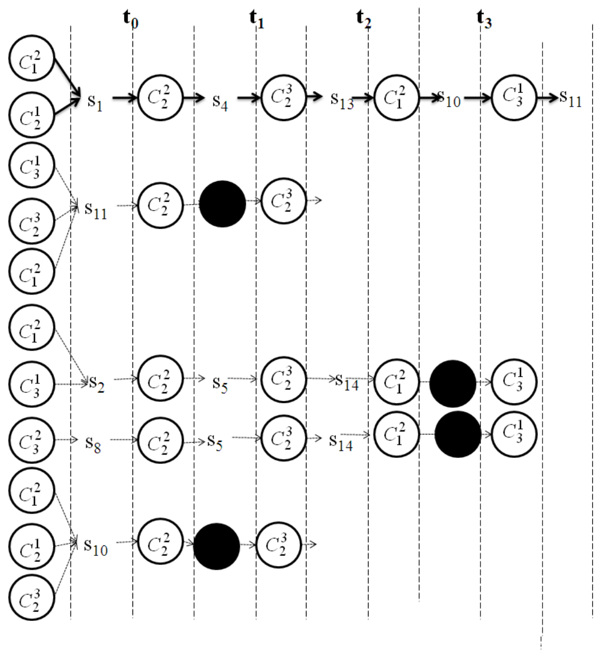

Figure 6. P^C consistent with the ω and BM

$D(t) = \{P^{C_1}, P^{C_2}\}$ = { $\{r_0(C_2^1, C_2^2, s_1), r_1(C_2^2, C_2^3, s_4), r_2(C_2^3, C_1^2, s_{13}), r_3(C_1^2, C_3^1, s_{10})\}$, $\{r_0(C_1^2, C_2^2, s_1), r_1(C_2^2, C_2^3, s_4), r_2(C_2^3,$

C_1^2, s_{13}), r_3(C_1^2, C_3^1, s_{10})}} and the state Path corresponding is S-Path = {s_1, s_4, s_{13}, s_{10}, s_{11}}. The interpretation of the results with the behavior model shows that the system passed from the ok mode (the grey states in Figure 4) : states (s_1, s_4) to leaking mode (the dark states in Figure 4) : states (s_{13}, s_{10}, s_{11}).

7 CONCLUSION

This paper proposes an algorithm to diagnose dynamic systems modeled with the TOM4D methodology according to the Theory of Timed observations of [8]. This alogorithm is a preliminary work since we have not exploited all the potentialities of the theory. In particular, this algorithm does not consider the lack of timed observations that can occur in large and complex systems. An extension is under consideration with the idea to use the function model $FM(Pr(t))$ to discriminate between a true lack and an inconsistency.
On other hand, with large and complex systems, the impossibility to define a global behavior model obliges to model the behavior in a decompositional way with the description of the behaviors of each component of the system. Another extension to the proposed algorithm aims at computing the diagnosis locally for each component before merging the local diagnosis to get a global diagnosis. In the D.E.S. approaches, the diagnoses are merged using the events which are common with the local diagnosis. According to the TOM4D methodology, the observations classes are not common between two components because, by construction, each variable x_i is associated with one and only one component c_i. Consequently, the idea is to use the functional model $FM(Pr(t))$ to define the relation between the observation classes and to merge the local diagnosis.

ACKNOWLEDGEMENTS

The authors would like to thank the Provence - Alpes - Côte d'Azur Région and FEDER (Fonds Européen de DEveloppement Régional) for their funding.

REFERENCES

[1] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.
[2] L. Chittaro, G. Guida, C. Tasso, and E Toppano, 'Functional and teological knowledge in the multi-modeling approach for reasoning about physical systems: A case study in diagnosis', in *IEEE Transactions on Systems, Man, and Cybernetics, 23(6)*, pp. 1718–1751, (1993).
[3] L. Console, C. Picardi, and M. Ribaudo, 'Diagnosing and diagnosability analysis using pepa', in *14th European Conference on Artificial Intelligence*, (2000).
[4] M.-O. Cordier and A. Grastien, 'Exploiting independence in a decentralised and incremental approach of diagnosis', in *20th International Joint Conference on Artificial Intelligence*, (2007).
[5] I. Fakhfakh, M. Le Goc, L. Torres, and C. Curt, 'Modeling and diagnosis characterization using timed observation theory', in *7th International Conference on Software and Data Technologies, ICSoft*, (2012).
[6] I. Fakhfakh, M. Le Goc, L. Torres, and C. Curt, 'Modeling dynamic systems for diagnosos: Pepa/tom4d comparison', in *14th International Conference on Enterprise Information Systems, ICEIS*, (2012).
[7] M. Le Goc, 'SACHEM, a real-time intelligent diagnosis system based on the discrete event paradigm', *Simulation*, **80**(11), 591–617, (2004).
[8] M. Le Goc, 'Notion d'observation pour le diagnostic des processus dynamiques : application sachem et à la découverte de connaissances temporelles', in *Université Aix-Marseille III - Faculté des Sciences et Techniques de Saint Jérôme*, (2006).
[9] M. Le Goc, E. Masse, and C. Curt, 'Modeling processes from timed observations', in *3rd International Conference on Software and Data Technologies, ICSoft*, (2008).
[10] R. Rosenberg and D. Karnopp, 'Introduction to physical system dynamics', in *McGraw-Hill, Inc. New York, NY, USA.*, (1983).
[11] Th. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van de Velde, and B. J. Wielinga, 'Publication, knowledge engineering and management the commonkads methodology', *MIT Press*, (2000).
[12] C. Zanni, M. Le Goc, and C. Frydmann, 'A conceptual framework for the analysis, classification and choice of knowledge-based system', in *International Journal of Knowledge-based and Intelligent Engineering Systems, 10*, pp. 113–138. Kluwer Academic Publishers, (2006).

Comparison of Distributed Diagnosis Methods on Networks with Different Properties

Priscilla Kan John[1] and **Alban Grastien**[2]

We conduct a performance comparison between two algorithms for distributed diagnosis of discrete event systems. We are particularly interested in the time and memory requirements as the shape and size of the system are varied. To this end, we consider different classes of systems as can be found in the complex network literature [3]. We analyse and compare how certain properties of the network affect the performance of the two diagnosers. Valuable insight is gained into how network characteristics affect diagnostic performance and allows us to choose appropriate strategies for electricity networks, which are our motivating application.

1 Introduction

Real-world systems are often distributed in nature, *i.e.* they consist of a set of interconnected components. While the behaviour of each individual component can be simple, their emergent global behaviour is complex. As such systems increase in size, their supervision become more and more challenging. Diagnosis is the process of determining what happened on a system, including the ability to detect faults on the system and if possible isolate the fault(s). Diagnosis is thus an important aspect of the supervision of systems to ensure their smooth running or to take appropriate remedial actions. As systems become more and more complex, new methods to handle diagnosis are required. In *model-based diagnosis* (MBD) – our focus – the reconstruction of what happened on a system starting from a given initial state, takes into account observations on the system and a given model of the considered system.

We take a *Discrete Event System* (DES) approach to MDB [2] and use the automaton formalism for implementation. Regardless of chosen implementation, the search space involved in the overall history reconstruction of a system as a monolithic block grows exponentially with the size of the system (*i.e.*, the number of components considered). Hence for large systems, the diagnostic task becomes intractable. Different approaches have been investigated to handle this problem where the overall global computation is avoided by calculating smaller 'local' diagnoses. Decentralised techniques [12, 4] still require a final merging operation of the local diagnoses and hence still suffer from computational blowup. Several distributed algorithms have bee developed that further limit the problem by avoiding the final merging step: CPLC [13] and JT [8].

These distributed algorithms are very sensitive to the shape of the network and the number of components it contains (its size). The shape of the network refers to how the individual components are connected to other components and is generally domain-dependent. For instance, power networks have been shown to exhibit a *small-world* structure [14]. In this paper, the performance of the two diagnosis algorithms (CPLC and JT mentioned above) are studied and evaluated over a number of representative classes of networks. To this end, we introduce a generic local diagnosis and vary the shape and size of the network.

This paper is divided as follows. In Section 2, we introduce the problem of diagnosis of DES, and the algorithms of interest. In Section 3, we introduce the classes of networks that we considered in this work, and their associated attributes. In Section 5, we present the results of our experiments.

2 Diagnosis of Discrete Event Systems

We now present the diagnosis of discrete event systems and the two existing algorithms that compute all the trace consistent with the observation that avoid computing the global diagnosis.

2.1 Languages

An *alphabet*, generally denoted Σ, is a finite set of *letters*. A *word*, generally denoted w, over alphabet Σ is a (possibly empty and always finite) sequence of letters of the alphabet: $w \in \Sigma^\star$. A language, generally denoted $\mathcal{L}$, over alphabet Σ is a (potentially infinite) set of words: $\mathcal{L} \subseteq \Sigma^\star$.

Let w be a word defined over alphabet Σ. The *projection* of w over alphabet $\Sigma' \subseteq \Sigma$, denoted $Proj_{\Sigma\to\Sigma'}(w)$ is the restriction of w to the letters of Σ'. This is formally computed by

$$Proj_{\Sigma\to\Sigma'}(w) \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \sigma w' & \text{if } w = \sigma w' \text{ and } \sigma \in \Sigma' \\ w' & \text{if } w = \sigma w' \text{ and } \sigma \notin \Sigma' \end{cases}$$

where ε represents the empty word. In general, the alphabet Σ will be ignored in the formulas. The projection over alphabet Σ' of language $\mathcal{L}$ defined over alphabet $\Sigma \supseteq$ is defined by the projection of each of its words: $Proj_{\Sigma'}(\mathcal{L}) = \{w' \in \Sigma'^\star \mid \exists w \in \mathcal{L}. w' = Proj_{\Sigma'}(w)\}$. The reverse operation computes all words on the latter alphabet whose projection on the former alphabet belong to the specified language:

$$Proj_{\Sigma\to\Sigma'}{}^{-1}(\mathcal{L}') = \{w \in \Sigma'^\star \mid Proj_{\Sigma\to\Sigma'}(w) \in \mathcal{L}'\}$$

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages defined respectively over alphabets Σ_1 and Σ_2. The synchronisation of both languages, denoted $\mathcal{L}_1 \otimes \mathcal{L}_2$, is defined as the set of words on the union of their alphabets whose projection on each alphabet belong to the corresponding language:

$$\mathcal{L}_1 \otimes \mathcal{L}_2 = \{w \in (\Sigma_1 \cup \Sigma_2)^\star \mid \forall i \in \{1,2\}. Proj_{\Sigma_i}(w) \in \mathcal{L}_i\}$$

[1] ANU, Australia
[2] NICTA and ANU, Australia

(Notice that the operation is associative and commutative.)

Finally, we present two operations of consistency. Given a set of languages $\{\mathcal{L}_1, \dots \mathcal{L}_k\}$ possibly defined on separate alphabets $\Sigma_1, \dots, \Sigma_k$, the global consistency of language $\mathcal{L}_i$ is the synchronisation of all those languages projected on Σ_i is $\mathcal{L}_i$.

$$gcons(\mathcal{L}_\rangle, \{\mathcal{L}_1, \dots \mathcal{L}_k\}) = Proj_{\Sigma \to \Sigma_i}(\mathcal{L})$$

where $\Sigma = \bigcup_{j \in \{1,\dots,k\}} \Sigma_j$ and $\mathcal{L} = \bigotimes_{j \in \{1,\dots,k\}} \mathcal{L}_j$. If the language $\mathcal{L}_i$ is unmodified by the operation of global consistency, we say that the language is globally consistent.

The *local consistency* is similar to the global consistency, except that it is performed pairwise:

$$lcons(\mathcal{L}_\infty, \mathcal{L}_2) = Proj_{\Sigma_1 \cup \Sigma_2 \to \Sigma_1}(\mathcal{L}_1 \otimes \mathcal{L}_2).$$

Practically, the language will be implemented by an automaton.

2.2 Diagnosis

In diagnosis, we consider a system consisting of a number of connected components; for this reason, it will be called a network.

A letter will represent an *event* that can occur on the network. A word, i.e., a sequence of event, is a *behaviour* of the network. A language is a set of possible behaviours.

The *model* $\mathcal{L}_M$ (defined on Σ) of the network is the set of behaviours that can physically take place in the network. When certain events occur, they are received by the monitoring system; these events, denoted $\Sigma_o \subseteq \Sigma$, are called *observable*.

A behaviour $w \in \mathcal{L}_M$ takes place in the network. This behaviour is unknown, but is partially observed. The observation is the restriction of w to the observable events, i.e., $obs = Proj_{\Sigma \to \Sigma_o}(w)$. The problem of diagnosis is finding w; because w is only partially observed, it is impossible to retrieve it precisely in general. Instead, the diagnosis is defined as the subset of the model that could produce the observation:

$$\Delta = \mathcal{L}_M \otimes \{obs\}.$$

2.3 Decentralised Diagnosis

In practice, the network is a set of interconnected components. This affects the diagnostic problem in two ways: the complexity of the problem and the observations.

The size of the automaton representing $\mathcal{L}_M$ is exponential in the number of components. Because the network can include hundreds or more components, this automaton cannot be computed. Instead, the network is represented by a set of component models $\{\mathcal{L}_{M1}, \dots, \mathcal{L}_{Mk}\}$ which implicitely represent the model $\mathcal{L}_M = \bigotimes_{i \in \{1,\dots,k\}} \mathcal{L}_{Mi}$. The events that belong to several components models are called synchronisation events.

Because of the physical decentralisation of the network, it is often impossible to determine precisely the order between the observations coming from different components [9]. The observation of the behaviour w is therefore the set of local observations: $\forall i \in \{1, \dots, k\}.obs_i = Proj_{\Sigma \to \Sigma_o \cap \Sigma_{Mi}}(w)$. Consequently, the observation of the trace w is a set of observations defined by the language: $\mathcal{L}_O = \bigotimes_{i \in \{1,\dots,k\}} \{obs_i\}$. We write $\{obs_i\} = \mathcal{L}_{Oi}$.

A decentralised diagnosis is a representation of the diagnosis as a set of languages whose synchronisation is the diagnosis. For instance, the following set is a decentralised diagnosis:

$$\{\mathcal{L}_{M1}, \dots, \mathcal{L}_{Mk}, \mathcal{L}_{O1}, \dots, \mathcal{L}_{Ok}\}.$$

Not all decentralised diagnosis are equal. In order to provide useful information, a decentralised diagnosis must be globally consistent. A globally consistent diagnosis means that each local diagnosis is the precise restriction of the global diagnosis on a set of events; for instance, if a particular event e is of interest, and the local diagnosis $\mathcal{L}_i$ is defined over an alphabet that include e, then e possibly (resp. surely) occurred iff a trace (resp. all traces) of $\mathcal{L}_i$ contains the event e.

We want to compute the global consistency of the decentralised diagnosis $\{\Delta_1, \dots, \Delta_k\}$ where for each i, $\Delta_i = \mathcal{L}_{Mi} \otimes \mathcal{L}_{oi}$. This can be done by computing the global diagnosis $\Delta = \bigotimes_{i \in \{1,\dots,k\}} \Delta_i$ and projecting it back to the local alphabets. However, computing Δ is generally impractical. The two algorithms presented in the following two subsections aim at computing the globally-consistent decentralised diagnosis without computing Δ. In practice, we will only compute the global consistency of one local diagnosis which we will arbitrarily choose to be Δ_k.

2.4 CPLC

The Computational Procedure for Local Consistency (CPLC, [13]) incrementally synchronises the local diagnoses and projects the diagnosis on the set of events of the remaining local diagnoses, hence reducing the complexity.

Let $J \subseteq I = \{1, \dots, k\}$ be a subset of local diagnoses and $\Sigma_J = \bigcup_{i \in J} \Sigma_i$. CPLC inductively compute the following languages:

- $M_1 = \Delta_1$,
- $W_i = Proj_{\Sigma_{\{1,\dots,i\}} \cap \Sigma_{\{i,\dots,k\}}}(M_i)$,
- $M_{i+1} = W_i \otimes \Delta_{i+1}$.

It can be proved that $Proj_{\Sigma_k}(\mathcal{M}_k)$ is the globally consistent version of Δ_k. The computation of Δ is avoided by the projection in the second above. CPLC also implements a one-step lookahead in an attempt to find the optimal sequence of local diagnoses.

2.5 Junction Tree Algorithm

A nice method to improve the precision of a decentralised diagnosis is to perform local consistencies (LC). LC does not require to compute the global diagnosis. In general, however, LC does not ensure global consistency (GC). The equivalence between LC and GC has been proved [13] in certain circumstances: two local diagnoses share a connection if their alphabets intersect. Consider the graph of local diagnoses where the nodes of the graph are the local diagnoses and the edges are the connections between the local diagnoses. If this graph forms a tree (or a forest), then LC implies GC.

It is rarely the case that the network forms a tree. However, it is possible [7] to transform any network in a tree where each node is a clique, i.e., a set of nodes from the original graph. This transformation is called a *junction tree*. An important measure of a junction tree is the *tree width*, i.e., the size of the biggest clique. The optimal tree is the one that minimises the tree width, and the tree width of the original graph is the tree width of its optimal junction tree.

Diagnosis of DES by junction tree [8] transforms the network in a junction tree, computes the local diagnosis of each clique, and perform the local consistency, starting from the leaves of the tree and ending with the root. The root is then globally consistent.

3 Networks

A network in the physical sense is a set of interconnected components. The most flagrant example is the Internet where computers are linked together through telecommunication channels. The electricity grid is also another example of an important network that connects generators, transformers and loads. At the abstract level, a network is a *graph* in the mathematical sense [1].

Definition 3.1 (Graph)**.** A graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ consists of a set of vertices $\mathcal{V}$ (also called *nodes*) linked together by a set of edges $\mathcal{E}$. Each edge $edg \in \mathcal{E}$ is connected to a pair of vertices (ver_i, ver_j) one on each side.

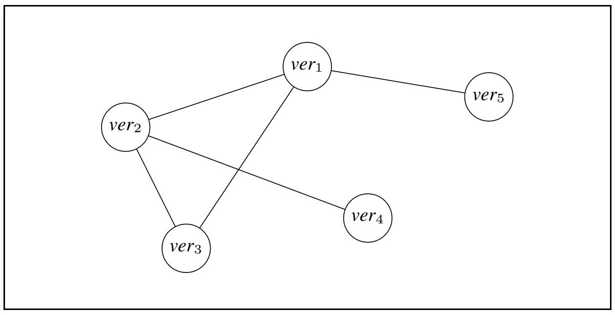

Figure 1. Example of a network with 5 vertices and 5 edges

The components of a network are represented as the *vertices* of the graph and they are connected by *edges* according to some defined rules. A network has an associated physical *topology* (structure), which describes which components is connected to which other components. In the networks we deal with, it is assumed that no self-loop exist and that the representative graphs are undirected with no repeated edge. An example of a graph with five vertices is shown in Figure 1.

3.1 Network Metrics

To compare inherent characteristics of various types of networks, we need a set of metrics that is able to capture and quantify certain properties that are displayed by the networks. We describe three such metrics in this section.

3.1.1 Average Path Length

To explain what is meant by *average path length*, we first need to define the concept of *distance* between two vertices.

Definition 3.2 (Distance between two vertices)**.** The *distance* d_{ij} between two vertices labelled ver_i and ver_j is given by the total number of edges that connect them through shortest linkages.

For example, for the graph represented in figure 1, the distance d_{12} between vertices ver_1 and ver_2 is equal to 2. Similarly, $d_{12} = 1$ and $d_{45} = 3$.

Definition 3.3 (Average Path Length)**.** The average path length of a network is defined to be the average value of all distances over the network:

$$avepath = \frac{2}{N(N-1)} \sum_{i,j(i<j)} d_{ij} \quad (1)$$

where N is the *size* of the network, *i.e.* the total number of vertices in the network.

For the network shown in figure 1, the average path length is 1.6 given the respective distances between vertices as shown in table 1 :

Value	d_{ij}
1	$d_{12}, d_{13}, d_{23}, d_{24}, d_{15}$
2	$d_{14}, d_{34}, d_{25}, d_{35}$
3	d_{45}

Table 1. Distances in example network

The average path length of a network gives an idea of how easily nodes can be reached in the network. Generally, a short average path length is more desirable as it means information passing between nodes can be done faster.

3.1.2 Clustering Coefficient

The *clustering coefficient* of a network gives an idea of how isolated or connected the network is.

Definition 3.4 (Clustering Coefficient)**.** Let ver_i be a vertex in a network, where ver_i has k_i edges connecting it to k_i other vertices known as the *neighbours* of ver_i. The maximum number of edges among those k_i vertices is given by $k_i(k_i - 1)/2$. Let K_i be the actual number of edges existing between the k_i vertices. The *clustering coefficient* C_i of vertex ver_i is given by

$$C_i = \frac{2K_i}{k_i(k_i - 1)} \quad (2)$$

The *clustering coefficient* of the whole network is the averaged value of the clustering coefficients of all the vertices in the network ($0 \le C \le 1$). $C = 0$ if and only if all vertices in the network are isolated, *i.e.*, have neighbours that are not connected to each other, and $C = 1$ if each vertex is connected to every other vertex in the network. The clustering coefficient is a measure of how likely it is that the neighbours of a node in the network are connected to each other.

We once again consider the example network in Figure 1 to illustrate. Table 2 shows the clustering coefficient values for the vertices shown in Figure 1.

Vertex	Number of neighbours	K_i	C_i
ver_1	3	1	$\frac{1}{3}$
ver_2	3	1	$\frac{1}{3}$
ver_3	2	1	1
ver_4	1	0	0
ver_5	1	0	0

Table 2. Clustering coefficients in example network

Hence, the clustering coefficient C of the network, given by the average of all clustering coefficients in Table 2 is equal to $(\frac{1}{3} + \frac{1}{3} + 1 + 0 + 0)/5 = \frac{1}{3}$.

3.1.3 *Degree and Degree Distribution*

We define here the concept of the *degree* of a vertex in an *undirected* network.

Definition 3.5 (Degree)**.** The degree of a vertex ver_i in an undirected network is the number k_i of the edges connecting it to its k_i neighbours.

In a *directed* network, we distinguish between incoming edges and outgoing edges of a node. But this issue is irrelevant for undirected networks.

Intuitively, a vertex of higher degree will have more significant influence on the network because it is involved in more connections. The *average degree* $\langle k \rangle$ of a network is the average value of vertex degrees over the entire network.

In a network, every vertex has a degree value, some large and some small. The distribution of vertices of certain degree could be of interest to better understand the network. This distribution is called the *degree distribution* of the network.

Definition 3.6 (Degree Distribution)**.** The degree distribution of a network is defined by a probability function *Prob* (k), which is the probability that a randomly picked vertex will have degree k, assuming each vertex has equal probability to be picked (uniform distribution).

Once more, we use the example given in Figure 1 to illustrate. The degrees of each vertex is equivalent to its number of neighbours and hence is given by Column 2 in Table 2. The average degree $\langle k \rangle$ of the network is equal to ($\frac{3+3+2+1+1}{5}$) = 5.

$\boldsymbol{k}$	*Prob* $(\boldsymbol{k})$
1	$2/5$
2	$1/5$
3	$2/5$

Table 3. Degree distribution in example network

3.2 Network Configurations

Next, we describe a few different types of physical topology for networks. These topologies are used as starting structures for systems on which we test the performance of the two distributed diagnosis algorithms. Understanding the relationship between the topology of networks and the performance of our algorithm provides insight into how best to design networks or how best to tackle operations on networks given their structures. We explore two types of *regular* networks (branching and ring networks) and two types of randomised networks (random networks and small world networks). *Regular* networks are constructed in a predictable way whereas randomised networks have an element of randomness in their structure.

For comparison purposes, we consider a *fully connected* network. A *fully connected* network is a type of regular network where there exists an edge between any pair of vertices in the network. It has an average path length given by $avepath_{full} = 1$ and a clustering coefficient given by $C_{full} = 1$, both values being self-evident from their respective definitions 3.4 and 3.3. The total number of edges in a fully connected network of size N (*i.e.* with N vertices) is $\frac{N(N-1)}{2}$ [3]. We note that among all networks with the same number of vertices, the fully connected network has the shortest average path length of 1, and the biggest clustering coefficient of 1.

3.2.1 *Ring Network*

In a ring network of size N, each node is connected to $2K$ nearest-neighbours, where $K > 0$ is an integer. A ring network with $K = 2$ is shown in Figure 2.

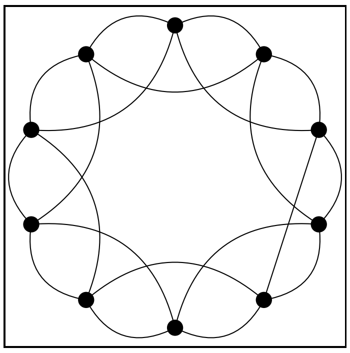

Figure 2. Example of a ring network with ten vertices and $K = 2$

The clustering coefficient for a ring network is given by

$$C_{ring} = \frac{3(K-1)}{2(2K-1} \tag{3}$$

For large K ($K \to \infty$), $C_{ring} \to \frac{3}{4}$.

The average path length of a ring network, given M number of edges in the network, is given by

$$\overline{L}_{ring} = \frac{M(M+1) - 2(K-1)(M-K+1)}{2M} \tag{4}$$

For large M ($M \to \infty$), $\overline{L}_{ring} \to \infty$. Derivations for C_{ring} and $\overline{L}_{ring}$ can be found in [3].

3.2.2 *Branching Network*

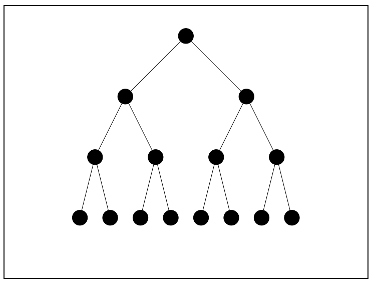

Figure 3. Example of a branching network with $n = 3$ and $r = 2$

A branching network has a tree structure where the vertices of the tree are the actual nodes of the network. It can further be described as an *r-ary* tree whereby each vertex which is not a leaf has exactly r children. A 2-ary tree is more commonly known as a binary tree and a 3-ary tree as a ternary tree. We consider the root of the tree to be of *order* 0, and we write $\rho = 0$. The *tree order* is the partial ordering on the vertices of a tree with $u \leq v$ if and only if the unique path from the root to v passes through u. The leaf vertices are considered to be at layer η ($\rho = q$). An example of a branching networks is shown in Figure 3.

The clustering coefficient for a branching network is given by $C_{branch} = 0$, self-evident from definition 3.4.

Let ρ represent a layer in a branching network. The number of edges found in layer ρ is given by $r^{\rho+1}$. The average path length for a branching network is given by

$$\overline{L}_{branch} = \frac{1}{(r-1)(r^{q+1}-1)} \sum_{\rho=0}^{q-1} r^{\rho+1}\left(r^{q-\rho}-1\right)\left(r^{q+1}-r^{q-\rho}\right) \tag{5}$$

(Proof omitted due to space restrictions.) From equation 5, it is interesting to note that for $q = 1$, *i.e.* for a star network, as $r \to \infty$, $\overline{L}_{branch} \to r$.

3.2.3 Random Network

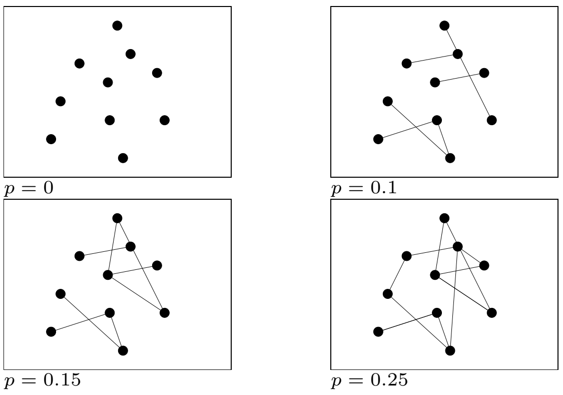

Figure 4. Example of random networks with 10 vertices as p is increased

A random network is the opposite extreme of a regular network. There is no predetermined structure to the graph and vertices are connected with an element of randomness. The random network was introduced by Erdös and Rényi in their seminal papers [5, 6]. A random network consists of N vertices joined by edges which are placed between pairs of vertices chosen uniformly at random. Every possible edge between any two vertices is present with probability p from a uniform distribution, and absent with probability $1 - p$. To construct a random network, we start with N isolated vertices, where $N \gg 1$. Two vertices are randomly picked and then connected by an edge with probability p from a uniform distribution. The vertices are then put back to the pool and the process is repeated for a suitably large number of steps. The procedure will yield approximately $pN(N-1)/2$ number of edges. Figure 4 illustrates some random graphs generated from 10 isolated vertices with different values of p.

A significant result in [6] is that important properties of random graphs emerge at certain values of p. For example, when p is larger than a certain threshold $p_c \sim \frac{\ln N}{N}$, almost all random graphs generated in the above described way will be connected, whereas for $p < p_c$, almost all graphs generated were not connected networks (*i.e.* contain isolated clusters).

The average degree of a random network of size N is given by

$$\textit{avedegRand} = p(N-1) \tag{6}$$

The average path length satisfies

$$\overline{L}_{rand} \sim \frac{\ln N}{\ln \textit{avedegRand}} \tag{7}$$

The clustering coefficient is given by

$$C_{rand} = p \approx \frac{\textit{avedegRand}}{N} \tag{8}$$

The degree distribution in a random network follows a *Poisson distribution*:

$$\textit{Prob}\,(k) = \frac{\mu^k}{k!}\exp^{-\mu} \tag{9}$$

where μ is the expectation value, $\mu = pN \approx \textit{avedegRand}$. The proofs for equations (9) to (8) can be found in [3].

We note that when N is large, such random networks may have a relatively small average path length because the growth of $\ln N$ is much slower than that of N in equation (7). Also from equation (8), a large scale random network (large N) does not have prominent clustering features.

3.2.4 Small-World Network

As mentioned previously, a ring network has a high clustering coefficient and a large average path length. On the other hand, a random network possesses a small clustering coefficient and displays short average path length. There is a category of networks that is characterised by both a high clustering coefficient and a short average path length. This category is known as the *small-world* networks. They were introduced by Watts and Strogatz in [14]. Since then, there have been modified versions of th algorithm presented in [14] but we will stick to the original version in this thesis when referring to *small-world* networks.

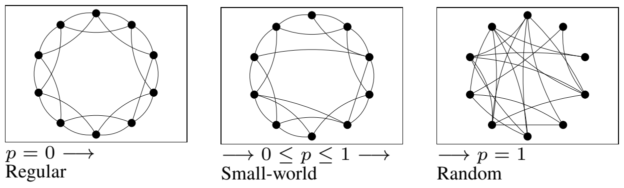

Figure 5. Transition from regular to small-world to random networks as the value of p increases

A small-world network can be generated as follows [14]:

1. We start with a ring network with N vertices, each connected to $2K$ nearest neighbours.
2. We choose a vertex and the edge that connects it to a nearest neighbour in a clockwise (or anti-clockwise) sense.
3. This edge is reconnected with probability p_{sw} to a vertex chosen uniformly at random over the entire ring, with duplicate edges forbidden; otherwise we leave the edge alone (with probability $1 - p_{sw}$.
4. This process is repeated by moving clockwise (or anti-clockwise) around the ring, considering each vertex in turn until one lap is completed.
5. We then consider edges that connect vertices to their second nearest neighbours clockwise (or anti-clockwise). These edges are also rewired as before with probability p_{sw}.

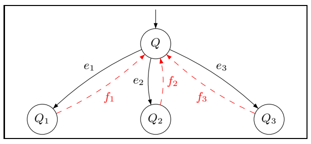

Figure 6. Automaton for a component with three neighbours

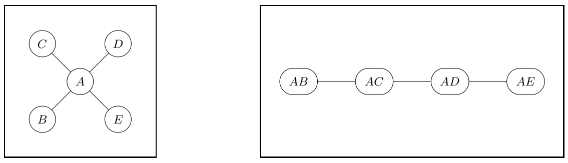

Figure 7. Example of a star network (left) and its junction tree (right)

6. We continue to circulate around the ring and proceed outward to more distant neighbours after each lap until every edge in the network has been considered once.

Since there are NK edges in the network, the rewiring process stops after K laps. Three examples generated using this process, with different values of p_{sw}, are shown in figure 5. It can be observed that for the case where $p_{sw} = 0$, the original ring stays unchanged. As the value of p_{sw} increases, the graph becomes increasingly disordered until for $p_{sw} = 1$ where all edges are rewired randomly and a random network (as described in section 3.2.3 is obtained. One main result of [14] is that for $0 < p_{sw} < 1$, the graph displays *small-world* properties, *i.e.* high clustering yet small average path length.

For large enough size N, the clustering coefficient of the small-world network is given by

$$C_{sw}(p) = \frac{3(K-1)}{2(2K-1)}(1-p)^3 \qquad (10)$$

The average path length can be expressed as

$$\overline{L}_{sw}(p) = \frac{2N}{K} f(2Np/K) \qquad (11)$$

where

$$f(x) = \begin{cases} c, & x \ll 1 \\ \frac{\ln x}{x} & x \gg 1 \end{cases} \qquad \text{(typically } c = 1/4\text{)} \qquad (12)$$

The proof for equations 10 and 11 can be found in [10].

4 Experimental Setup

We implemented the distributed diagnosis algorithm JT [8] in **Java** and compared it to CPLC [13]. Experiments were run on an Intel Core 2D, 2.4 GHz, 2GB Linux machine.

Similarly to Su and Wonham's experiments [13], we started with generic local diagnoses. Essentially, each component may communicate with any of its neighbours; when the communication takes place, the two components must first exchange the message e_i followed by the message f_i. Only after the communication has ended, can the two components communicate with other components. This is illustrated in Figure 6 for a component with three neighbours. For k neighbours, a local diagnosis contains $k+1$ states and $2 \times k$ transitions. This automaton was chosen because it exhibits some synchronisation and some concurrency.

5 Results

In this section, we present the results of the performance of our distributed diagnosis algorithm on a range of different graphs presented in section 3 as their size and other properties are varied.

5.1 Results on Branching Networks

Table 4 shows results obtained on branch networks with varying parameter values for q and r. We note that a star network is a special case of a branching network where $q = 1$. The size of the largest structure (synchronisation automaton) and the time for producing a diagnostic result were recorded for each network for two cases: one using CPLC, and two using JT.

q	r	n	CPLC		JT	
			largest struc	time (s)	largest struc	time (s)
1	5	6	[7, 12]	0.115	[37, 122]	0.103
1	10	11	[8, 24]	0.463	[122, 442]	0.220
1	15	16	[17, 32]	1.174	[257, 962]	0.340
1	20	21	[16, 64]	3.132	[442, 1682]	0.417
1	25	26	[16, 64]	3.053	[667, 2602]	0.898
1	30	31	[16, 64]	5.210	[962, 3722]	1.566
1	40	41	[32, 160]	17.91	[1682, 6562]	4.150
1	50	51	[32, 160]	44.36	[2602, 10202]	10.11
2	2	7	[4, 6]	0.127	[10, 26]	0.157
3	3	40	[96, 512]	7.339	[17, 50]	0.362
4	2	31	[54, 270]	5.097	[10, 26]	0.091
4	3	121	[576, 4416]	1103	[17, 50]	1.324
4	5	781	[640, 5504]	na	[37, 122]	216.9
5	3	364	[256, 2048]	na	[17, 50]	18.83
6	2	127	[1458, 13122]	5820	[10,26]	1.415
7	2	255	[1296, 12096]	40488	[10,26]	6.148
8	2	511	[4374, 45198]	na	[10, 26]]	44.14

Table 4. Results on branching networks

A few interesting points emerge from the results on branching networks. Overall, JT performed better than CPLC, especially when the network contains a large number of components. For networks containing around 10 components or less, the difference in the diagnosis time between the two methods is negligible although the largest structure in the JT method has a greater size.

For star networks ($q = 1$), the size of the largest structure is significantly larger in the JT method than in CPLC although the diagnosis time is better in the former. This can be explained by the fact that in CPLC, the 'junction line' is built on the graph of events rather than the graph of components and a heuristic ordering procedure is used to minimise the size of the synchronous automaton at every step. The star shape of the network also means that the root node appears in every cluster in the junction tree (figure 7), forcing multiple redundant synchronisation with the root node during diagnosis. Because the root node is the highest connected node, this creates large synchronising structures. The number of states in the largest synchronising structure in a star network is actually given by $n^2 + 1$.

Interestingly, for cases where $q > 1$, the size of the largest structure is more reasonable and depends on the value of the branching factor r only (and not on q). This makes sense since r controls the tree-width of the junction tree. On the other hand, q controls the number and length of branches in the junction tree. As q increases, for a fixed value of r, diagnosis time increases exponentially. Figure 8 which shows the diagnosis time against different values of q

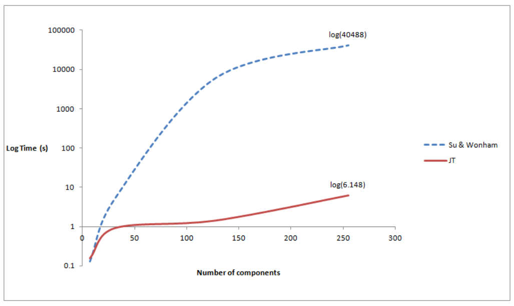

Figure 8. Diagnosis time(s) versus number of components for a branch network configuration ($r = 2$)

for $r = 2$. For both algorithms, the diagnosis time increases exponentially with the value of q, but diagnostic performance of the JT algorithm is better by orders of magnitude.

In the case of CPLC, the size of the largest structure quickly becomes unmanageable as q and r are increased. There are cases where memory is exhausted before the computation is finished and the largest structures reached are shown in red in table 4. In contrast, for JT, the largest structure remains manageable and retains a fixed value that dependents only on r and not on q.

5.2 Results on Ring Networks

Table 9 shows results obtained on ring networks with varying parameter values for n and k.

n	k	CPLC		JT	
		largest struc	time (s)	largest struc	time (s)
10	1	[4, 8]	0.264	[15, 50]	0.071
10	2	[36, 168]	0.893	[415, 2586]	0.916
10	3	[576, 4416]	11.13	[5714, 46954]	46.63
10	4	[6912, 69120]	497.7	[17148, 148896]	na
20	1	[4, 8]	0.459	[27, 108]	0.098
20	2	[4, 8]	3.447	[528, 3470]	1.664
20	3	[576, 4416]	598.8	na	na
30	1	[4, 8]	0.815	[27, 108]	0.324
30	2	[36, 168]	7.873	[528, 3470]	2.770
50	1	[4, 8]	0.815	[27, 108]	0.451
50	2	[36, 168]	20.00	[528, 3470]	6.360
100	1	[4, 8]	4.29	[27, 108]	0.749
100	2	[36, 168]	92.48	[528, 3470]	15.76
150	1	[4, 8]	5.452	[27, 108]	1.831
150	2	[36, 168]	203.4	[528, 3470]	24.21
200	1	[4, 8]	8.707	[27, 108]	4.035
2047	1	[4, 8]	2429	[27, 108]	4003

Table 5. Results on ring networks

The results show that for small values of k ($k < 3$), JT shows better diagnostic time in general except when n is really large. However, when $k > 3$, CPLC produces faster diagnostic time. The largest structure obtained with JT has more states and events than the equivalent one obtained with the CPLC method in every ring network we tested. In fact, in contrast to what is observed for branching networks, the size of the largest structure in the CPLC method stays relatively small. Moreover, where the computation exhausts memory for the JT algorithm, CPLC is still able to produce a result without running out of memory despite that the computation time is large. We also note that the size of the largest structure shows a dependence on the value of k, and not of n, both for the JT method and CPLC's method.

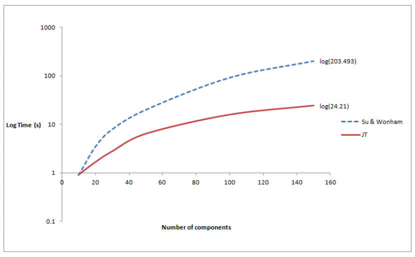

Figure 9. Diagnosis time(s) versus number of components for a ring network configuration ($k = 2$)

Figure 9 shows the diagnosis time for different values of n when $k = 2$. In both cases, the diagnosis time increases exponentially with the value of n although at a slower rate than for the branching networks in figure 8.

The results suggest that a combination of both methods, *e.g.* building a junction tree on the events of the system and using a heuristic ordering strategy for synchronisation, might further improve results on large networks that are highly connected.

5.3 Results on Random and Small-World Networks

Table 9 shows results obtained on small-world networks with varying parameter values for n_{ring}, k and p (the seed value used to generate p is kept constant with value 5 for all networks.

The number of starting components connected into a ring network is given by n_{ring}. After applying the small-world transformation where connections are rewired with probability p, the actual number of components that are connected might be different. As it turned out for the cases presented in table 6, all the resulting small-world networks except one (the one with $n_{ring} = 200$) kept the same number of components as the starting ring networks they were built from. Random networks are special cases of small-world networks where $p = 1$.

n_{ring}	k	p	CPLC		JT	
			largest struc	time (s)	largest struc	time (s)
10	1	0.1	[4, 8]	0.204	[24, 92]	0.043
10	1	0.5	[16, 64]	0.185	[12, 32]	0.029
10	1	1	[6, 14]	0.145	[7, 16]	0.047
20	1	0.1	[8, 24]	0.264	[26, 96]	0.052
20	1	1	[8, 24]	0.300	[217, 56]	0.097
50	1	0.1	[16, 64]	2.256	[30, 118]	0.297
50	1	0.2	[8, 24]	1.869	[29, 112]	0.313
50	1	0.3	[16, 64]	2.605	[33, 134]	0.204
50	1	0.4	[16, 64]	3.251	[25, 90]	0.326
50	1	0.5	[8, 24]	3.251	[25, 90]	0.326
50	1	0.6	[48, 256]	5.182	[43, 174]	0.223
50	1	0.7	[24, 104]	2.767	[32, 128]	0.264
50	1	0.8	[72, 408]	3.490	[21, 76]	0.205
50	1	0.9	[54, 270]	7.187	[17, 56]	0.239
50	1	1	[48, 256]	5.009	[13, 34]	0.204
100	1	0.1	[16, 64]	5.961	[36, 150]	1.061
100	1	0.2	[16, 64]	5.074	[24, 92]	0.554
100	1	1	[1024, 10240]	3295	[13, 36]	0.526
200#	1	0.1	[8, 24]	2.323	[21, 76]	0.254

# actual number of components after reconnection is 50

Table 6. Results on random and small-world networks

Both algorithms performed very well on random and small-world networks with $k = 1$, although the JT method had faster diagnosis time. We note that CPLC took a very long time to run for the case

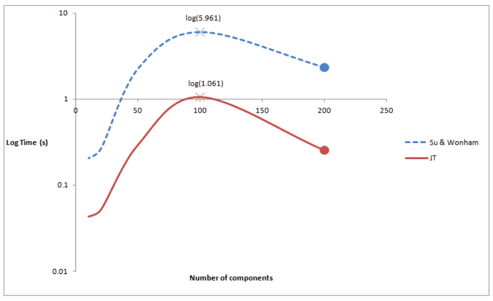

Figure 10. Diagnosis time(s) versus number of components for a small-world network configuration ($p = 0.1$)

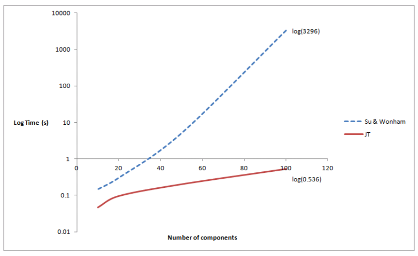

Figure 11. Diagnosis time(s) versus number of components for a random network configuration (small-world algorithm with $p = 1$)

where ($n_{ring} = 100, k = 1, p = 1$) whereas the JT method completed in under a second. This is due to the fact that the structure of the network is very close to a tree structure and the diagnosis result obtained on it shows similarity with a case in a branching network ($q = 6, r = 2$) with roughly the same number of components (see table 4).

Figure 10 shows the diagnosis time for small-world networks for the case where $k = 1$ and $p = 0.1$ using both the JT method and CPLC. The diagnostic time for both methods was very good, even for networks with a large number of components. The graphs dip when $n_{ring} = 200$ because the number of components that are actually connected in the network is 50. The algorithm employed to generate small-world networks is stochastic. It so happened that every other network generated retained a number of connected components equal to n_{ring}.

Both methods do not cope well when $k > 1$; memory is exhausted before the algorithm terminates. Given that small-world networks can have characteristics of both branching networks (near-zero clustering coefficient) and of ring networks (high clustering coefficient), when they display a fairly equal balance of both characteristics, either diagnosis method struggles. This suggests an approach that applies CPLC to parts of the network that have loops, and the JT method to parts that are tree-like. Investigating how to implement such an approach is material for future work.

6 Conclusion

We studied networks of different shapes and sizes and assessed the performance of the JT and CPLC diagnosis algorithms on them. It was confirmed that the shape of the network, defined by the relevant network parameters, has a significant impact on performance. For example, JT performs well on branching networks whereas CPLC performs well on ring networks. Randomised networks can exhibit characteristics of both branching and ring networks. Electricity networks (our motivating application) display small-world characteristics [14] but with a very near-tree structure by virtue of their design. Hence use of the JT algorithm for diagnosis on such networks is justified. For cases where portions of the network display high clustering (near-ring structure), we could apply CPLC. The best way to implement this dual-approach is an interesting area of future work. Another option to deal with near-ring portions of the network is to selectively ignore some of the connections on the network when performing diagnosis without compromising accuracy of the results [11].

REFERENCES

[1] V.K Balakrishnan. *Graph Theory*. McGraw-Hill, 1997.
[2] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
[3] G.R Chen. Complex networks: Modeling, dynamics and control. Lecture Notes for EE6605, 2009.
[4] M.-O. Cordier and A. Grastien. Exploiting independence in a decentralised and incremental approach of diagnosis. In M. Veloso, editor, *Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 292–297. AAAI press, 2007.
[5] P. Erdós and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290 – 297, 1959.
[6] P. Erdós and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17 – 61, 1960.
[7] F.V. Jensen and F. Jensen. Optimal junction trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, Seattle, Washington*, 1994.
[8] P. Kan John and A. Grastien. Local consistency and junction tree for diagnosis of discrete-event systems. In *European Conference on Artificial Intelligence (ECAI-08)*, Patras, Greece, 2008.
[9] G. Lamperti and M. Zanella. *Diagnosis of Active Systems*. Kluwer Academic Publishers, 2003.
[10] M.E.J. Newman and D.J. Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, December 1999.
[11] Y. Pencolé P. Kan John, A. Grastien and P. Ribot. Synthèse d'un diagnostiqueur distribué et précis. In *Reconnaissance des Formes et Intelligence Artificielle, Actes du 17 me congrs francophone AFRIF-AFIA (RFIA-10)*, Caen, France, 2010.
[12] Y. Pencolé and M.-O. Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence (AIJ)*, 164:121–170, 2005.
[13] R. Su and W. M. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *Transactions on Automatic Control*, 50(12):1923–1935, 2005.
[14] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393:440 – 442, June 1998.

Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Fault Diagnosis in Discrete-Event Systems: How to Analyse Algorithm Performance?

Yannick Pencolé [1]

Abstract. This paper addresses the problem of analyzing the performance of algorithms that solve the fault diagnosis problem in discrete-event systems in an experimental way. To achieve this purpose, we define a set of metrics for algorithm comparison on one hand. On the second hand, we propose an experimental platform based on a tool called DIADES (Diagnosis of Discrete-Event Systems) to run experiments and measure different algorithms.

1 INTRODUCTION

Fault Diagnosis in Discrete-Event Systems is the problem of determining the set of fault events that have occurred in a dynamic event-driven system usually based on a sequence of observed events. In the literature, there are many works and algorithms to solve this problem either from the AI community or the Automatic Control community. Most of these algorithms have been developed in order to solve special cases of system by introducing some assumptions or special properties (distributed systems [2, 12], coordinated systems [4], centralized diagnosis [15], decentralized diagnosis [12], distributed diagnosis [2], flexible diagnosis [11],...). The main drawback of these methods is not really in the methods themselves but is the lack of a generic method to experimentally compare these approaches to solve the same problems. The kind of question raised by this paper is: given one diagnosis problem, given the set of available algorithms that are able to solve this problem, which one is experimentally the best?

This is not the first time that this question has been asked: the DXC competition is definitely an attempt to answer the question [7]. However, in this paper, we adopt a complementary point of view. Instead of analyzing one real system as DXC proposes, we want to analyze algorithms on a specific class of systems (that is the class of discrete-event systems) independently from a model of any underlying and specific system. In the Model-Based diagnosis principle, there is a clear distinction between the Model and the algorithm that runs the Model [10]. Algorithm performance is affected by both the quality of the model and the algorithm itself, however the way to measure this performance is different. The quality of the model can be measured by correctness and diagnosability analyses whereas the intrinsic performance of an algorithm can be measured only by comparing it to the optimal algorithm on the *same* model whatever the quality of this model.

In order to find an answer to the previous question, it is first required to define what '*best*' means and a way to define why an algorithm would perform '*better*' than another one: this is the first contribution of this paper. We introduce a set of metrics for the specific problem of Fault Diagnosis in Discrete-Event Systems that can be exploited to compare the algorithmic performances. The second contribution of this paper is to propose a framework to define common problems (benchmarks) that can be freely utilized by the community. As opposed to many contribution on Model-Based diagnosis, we propose here to rely on randomly generated problems to measure the algorithm performance: as explained above, we are interested here in comparing the intrinsic performance of algorithms on the same model, whatever the quality of that model.

This paper is organized as follows. Section 2 first recalls the formal definition of the problem in the classical manner. Section 3 discusses the notion of performance of diagnosis algorithms that tend to solve the problem. Section 4 introduces some metrics whose objective is to measure the performance of algorithms. Section 5 describes some simple experiments, it firstly describes how the studied problem has been randomly generated by DIADES and secondly illustrates the proposed metrics by comparing the performance of two basic algorithms ('Forward breadth-first search' and 'Forward depth-first search').

2 FAULT DIAGNOSIS PROBLEM

This section formally recalls the fault diagnosis problem on dynamic discrete-event systems as defined for instance in [15].

2.1 Discrete-event system

A Discrete-Event System (DES) is a system whose behaviour is fully characterised at any time by a sequence of events that has occurred from an initial state. In this paper, the system is supposed to be a set of N components $\mathrm{Sys} = \{\mathrm{Comp}_1, \dots, \mathrm{Comp}_N\}$ that are communicating with each other by synchronized events. There are many formalisms to represent such a system ([15], communicating automata [9], process algebra [3], Petri nets [2]...). Without loss of generality, the formalism chosen in this paper is synchronized automata.

Definition 1 (Model of a component) *The model of a component* Comp_i *is an automaton* $\mathcal{A}(\mathrm{Comp}_i) = (Q_i, \Sigma_i, T_i, q_{0i})$ *where*

- Q_i *is a finite set of states;*
- Σ_i *is the finite set of events;*
- $T_i \subseteq Q_i \times \Sigma_i \times T_i$ *is the finite set of transitions;*
- q_{0i} *is the initial state.*

A *trace* $\tau = e_1 \dots e_k$ of a component Comp_i is a possible sequence of events such that there exists a finite transition path from state q_{0i} in the automaton $\mathcal{A}(\mathrm{Comp}_i)$: $q_{0i} \xrightarrow{e_1} q_1 \dots q_{j-1} \xrightarrow{e_j} q_j \dots q_{k-1} \xrightarrow{e_k} q_k$. Among the set of events Σ_i, two subsets are distinguished:

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France Univ de Toulouse, France, F-31400 Toulouse, France, email: yannick.pencole@laas.fr

1. Σ_i^{obs} is the set of observable events: an observable event, once it occurs, is supposed to be recorded by the diagnosis process;
2. Σ_i^{int} is the set of interacting events: if it occurs, an interacting event occurs at least in two components simultaneously and thus represents a communication between these two components.

No assumption is made about whether an interacting event is observable or not, however, for the sake of simplicity here as it is not the scope of the paper, we suppose that any interacting event e between Comp_i and Comp_j ($e \in \Sigma_i^{int} \cap \Sigma_j^{int}$) that is observable on Comp_i ($e \in \Sigma_i^{obs}$) is also observable on Comp_j ($e \in \Sigma_j^{obs}$). Any interacting event e is associated to a set of components that simultaneously interact each other so any interacting event implicilty represents a structural *connection* between these components. A connection between a set of components $\{\text{Comp}_{j_1}, \dots, \text{Comp}_{j_l}\}$ exists iff there exists an interacting event e that belongs to every component of this set and only to these ones. The set of connections between the components is called the *structural model* of the system, also called the *topology* . The model of the system results from the synchronised product of the component model.

Definition 2 (Model of a system) *The model of the system is the automaton $M = \mathcal{A}(\text{Sys}) = (Q, \Sigma, T, q_0)$ where*

- *$Q \subseteq Q_1 \times Q_N$ is the finite set of global states;*
- *$\Sigma = \bigcup_{i=1}^{N} \Sigma_i$ is the finite set of events;*
- *$T \subseteq Q \times \Sigma \times Q$ is the finite set of global transitions;*
- *$q_0 = (q_{01}, \dots, q_{0N})$ is the global initial state.*

The $(q_1, \dots, q_N) \xrightarrow{e} (q'_1, \dots, q'_N)$ belongs to T iff

1. e is not an interacting event and there exists one and only one $i \in \{1, \dots, N\}$ such that $q_i \xrightarrow{e} q'_i \in T_i$ and for any $j \in \{1, \dots, N\} \setminus \{i\}$ $q_j = q'_j$;
2. e is an interacting event and belongs to the components $\{\text{Comp}_{i_1}, \dots, \text{Comp}_{i_k}\}$ and for any $j \in \{i_1, \dots, i_k\}$ $q_j \xrightarrow{e} q'_j \in T_j$ and for any $j \in \{1, \dots, N\} \setminus \{i_1, \dots, i_k\}$,$q_j = q'_j$.

Finally, in the following Q and T will respectively denote the set of global states and transitions that are reachable from the initial state q_0. Straightforwardly, Σ^{obs} denotes the set of observable events $\bigcup_{i=1}^{N} \Sigma_i^{obs}$ and a *global trace* $\tau = e_1 \dots e_k$ of the system Sys is a possible sequence of events such that there exists a finite transition path from state q_0 in M: $q_0 \xrightarrow{e_1} q_1 \dots q_{j-1} \xrightarrow{e_j} q_j \dots q_{k-1} \xrightarrow{e_k} q_k$.

2.2 Diagnosis problem and solution

Now we are ready to define the classical Fault diagnosis problem on DES.

Definition 3 (Fault) *A* fault *is a non-observable event $f \in \Sigma$.*

A fault is represented as a special type of non-observable event that can occur on the underlying system. Once the event has occurred, we say that the fault is *active* in the system, otherwise it is *inactive*.

Definition 4 (Diagnosis problem) *A diagnosis problem is a triple* $(\text{SD}, \text{OBS}, \text{FAULTS})$ *where* SD *is the global model of a system* Sys, OBS *is the set of timed observations $\{(o_1, t_1), \dots, (o_m, t_m)\}$ with $t_i \in \mathbb{R}^+, t_i < t_{i+1}$, $o_i \in \Sigma^{obs}$, and* FAULTS *is the set of fault events defined over* SD.

Informally speaking, $(\text{SD}, \text{OBS}, \text{FAULTS})$ represents the problem of finding the set of active faults from FAULTS that has occurred relying on the model SD and the sequence of observations OBS.

Definition 5 (Diagnosis Candidate) *A diagnosis candidate is a couple (q, F) where q is a global state of M ($q \in Q$) and F is a set of faults.*

A diagnosis candidate represents the fact that the underlying system is in state q and the set F of faults has occurred before reaching state q.

Definition 6 (Solution Diagnosis) *The* solution Δ *of the problem* $(\text{SD}, \text{OBS}, \text{FAULTS})$ *is the set of diagnosis candidates (q, F) such that there exists for each of them at least one global trace τ of* SD *such that:*

1. *the observable projection of τ is exactly the sequence $o_1 \dots o_m$ and the last event of τ is o_m;*
2. *the set of fault events that has occurred in τ is exactly F;*
3. *the final state of τ is q.*

Informally, candidate (q, F) is part of the solution if it is possible to find out in SD a behaviour of the system satisfying OBS which leads to the state q after the last observation of OBS and in which the faults F have occurred.

3 Diagnosis Algorithms

There exist many algorithms to solve the fault diagnosis problem presented above [15], [9], [18], [12], [16], [5], [6]. The purpose of some of them is to perform *off-line diagnosis* (also called *post-mortem diagnosis*): they consider OBS as a fixed set of observations when the algorithm starts and their objective is to find the solution of the problem. On the other side, some algorithms are so-called *on-line* in the sense that when the algorithm starts, the underlying system is operating, at a given time OBS is partly known and the purpose of the algorithm is not only to provide in the end the solution to the problem $(\text{SD}, \text{OBS}, \text{FAULTS})$ but also to provide through the operating time of the system some *diagnosis updates*. Depending on the type of algorithms used, some performance indicators defined below may be pertinent or not to measure the performance of an algorithm. Performance of on-line algorithms especially requires a set of performance indicators related to time and diagnosis update capabilities that are not pertinent in the case of off-line diagnosis algorithms. Before introducing the performance indicators, a set of definitions related to the result provided by any algorithms is required.

Definition 7 (Current Diagnosis) *The* current diagnosis *of an operating algorithm* ALG *at time t is the set of candidates that* ALG *provides at time t if it is available, or the most recent set of candidates provided by* ALG *before t.*

Usually, off-line algorithms provide two set of candidates. The first set is provided when the algorithm starts and usually consist of $(q_0, \varnothing)$ whereas the second set is provided when the algorithm ends. However, some off-line algorithms may be incremental and thus provide more updates. On-line algorithms, as opposed to off-line algorithms, must provide updated sets of candidates so the current diagnosis is changing over the time.

Definition 8 (Final Diagnosis) *The* final diagnosis *of an algorithm* ALG *is the* current diagnosis *when the algorithm ends.*

The final diagnosis of an algorithm is the final result and thus should provide the solution of the problem.

3.1 Off-line algorithm performance

Since the underlying system Sys is real and has really emitted the set of observations OBS, any diagnosis problem is associated to the real trace τ_{real} of the system and, as a consequence the real diagnosis of the system.

Definition 9 (Real Diagnosis) *The real diagnosis is the candidate* (q_{real}, F_{real}) *associated to the real trace* τ_{real}.

Determining (q_{real}, F_{real}) or at least F_{real} is the main objective of any diagnosis algorithms but to reach this objective, there are many obstacles. First, Definition 4 defines a model-based diagnosis problem that relies on the existence of a sound model SD so that τ_{real} is assumed to be represented in the model SD. However, in practice as described in [1], this may be not the case so the algorithm may not be able to determine τ_{real} as a possible trace which means that (q_{real}, F_{real}) may not be representable in SD at all.

Definition 10 (Sound problem) *The problem* $(\mathrm{SD}, \mathrm{OBS}, \mathrm{FAULTS})$ *is sound if* τ_{real} *is represented in the problem.*

In other words, if the problem is sound, the solution diagnosis contains the real diagnosis.

Definition 11 (Sound Algorithm) *The algorithm* ALG *is sound with respect to the problem* $(\mathrm{SD}, \mathrm{OBS}, \mathrm{FAULTS})$ *if the solution diagnosis is contained in the final diagnosis.*

A sound algorithm ALG always provides a superset of the solution diagnosis which guarantees that if the problem is sound, ALG provides the real diagnosis.

Definition 12 (Accurate Algorithm) *The algorithm* ALG *is accurate with respect to the problem* $(\mathrm{SD}, \mathrm{OBS}, \mathrm{FAULTS})$ *if the final diagnosis is exactly the solution diagnosis.*

An accurate algorithm is a particular sound algorithm that returns the smallest superset of the solution diagnosis. An accurate algorithm takes benefit of all the observations to refine the candidates and provide the smallest but still correct refinement.

Second, even if the real trace is represented in SD, the lack of observable events may lead to many ambiguities, which means that the diagnosability level of the algorithm is poor.

Definition 13 (Diagnosable problem) *The problem* $(\mathrm{SD}, \mathrm{OBS}, \mathrm{FAULTS})$ *is diagnosable if the solution diagnosis contains only one candidate.*

Definition 14 (Fault-Diagnosable problem) *The problem* $(\mathrm{SD}, \mathrm{OBS}, \mathrm{FAULTS})$ *is fault-diagnosable if the solution diagnosis contains only candidates as* $\{(q_1, F), \dots, (q_k, F)\}$.

In other words, if the problem is fault-diagnosable then the occurrence of the faults F are certain, only the state estimate is ambiguous.

Property 1 *If the problem is sound and diagnosable, the final diagnosis of any accurate algorithm on this problem is the real diagnosis.*

This property provides the skeleton for the design of performance indicators. An algorithm that optimally performs on a given problem $(\mathrm{SD}, \mathrm{OBS}, \mathrm{FAULTS})$ requires soundness and diagnosability analyses on one hand as proposed for instance in [15], [14], [13], [1] and accuracy analyses on the other hand, which is the main concern of this paper.

3.2 On-line algorithm performance

An on-line algorithm must perform as well as an off-line when it ends, which means that the above definitions and properties also hold for on-line algorithms. However, the second challenge of on-line algorithms is to propose diagnosis updates through time relying on a flow of observations that leads to the full sequence of OBS. In other words, on-line algorithms should solve a finite-set of diagnosis problems derived from the initial problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS})$. At time t_i, a new observation o_i is received and the on-line algorithm should then provide a diagnosis update at time t_i that is the solution of the problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS}_i)$ where $\mathrm{OBS}_i = \{(o_j, t_j) \in \mathrm{OBS} \wedge t_j \leq t_i\}$.

Definition 15 (Perfect On-Line Algorithm) *An on-line algorithm* ALG *is* perfect *iff* $\forall i \in \{1, \dots, |\mathrm{OBS}|\}$*, the current diagnosis of* ALG *at time* t_i *is the solution of* $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS}_i)$.

Such an algorithm is called *perfect* as there is no other algorithms that can perform better: at any time, it provides the solution for the available set of observations. Such an algorithm exists and is defined in [Sampath95] and is called the diagnoser. The diagnoser for a given system description SD and a given set of faults FAULTS is a deterministic finite-state machine obtained by a complete knowledge precompilation: each state of the diagnoser provides the current diagnosis and each transition is labeled with an observable event of the system. To solve the diagnosis problem with the diagnoser, the underlying algorithm simply builds the unique transition path from the initial state of the diagnoser (the initial diagnosis) associated to the given sequence of observations OBS, the state ending this path provides the final diagnosis. For the observation (o_i, t_i) is the sequence OBS, it is sufficient to trigger one transition to get a diagnosis update, which is in constant time and *almost* instantaneous. Moreover, by the construction of the diagnoser during the precompilation stage, any diagnoser state reached by a transition path from the initial state associated to the sequence of observations OBS_i is the solution of the problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS}_i)$.

Now, why not keeping using this diagnoser to solve any online diagnosis problem if it is perfect? The reason is that the precompilation may not be practically feasible: recalling that N is the number of components in the system, the size of the machine to build is in the worst case in $o(2^{2^N})$ and in $o(2^{|\mathrm{FAULTS}|})$.

4 PERFORMANCE METRICS

This section lists a set of performance metrics. These indicators are generic in the sense that most of them compare any type of algorithm results two by two. In particular, it does not necessarily rely on the real diagnosis that is usually unknown unless further analyses on the system are perform or the faulty scenario results from a simulation in which faults have been injected [7]. If the real diagnosis Δ_{real} is known then it can be utilized as any other diagnosis to measure the performance of the available algorithms. As detailed in this section, the proposed metrics, once they are applied on the real diagnosis Δ_{real}, are equivalent to some metrics also proposed in [7].

4.1 Time-Independent generic metrics

Time-Independent metrics take into account two diagnoses (a set of diagnosis candidates) Δ_1 and Δ_2 resulting from the diagnosis problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS})$ and provide some ways to compare them. Let $\Delta_1 = \{(q_{i_1}, F_{i_1}), \ldots, (q_{i_l}, F_{i_l})\}$ and $\Delta_2 = \{(q_{j_1}, F_{j_1}), \ldots, (q_{j_k}, F_{j_k})\}$ be two diagnoses on a given diagnosis problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS})$.

Definition 16 (Belief State Distance) *The Belief State Distance* $\mathrm{BSD}(\Delta_1, \Delta_2)$ *is:*

$$|\{q, \exists (q, F) \in \Delta_1 \setminus \Delta_2\}| + |\{q, \exists (q, F) \in \Delta_2 \setminus \Delta_1\}|.$$

The metrics BSD relies on the so-called Hamming distance: it counts the number of differences between the two belief states $\{q_{i_1}, \ldots, q_{i_l}\}$ and $\{q_{j_1}, \ldots, q_{j_k}\}$ so it behaves as a distance that is null iff $\{q_{i_1}, \ldots, q_{i_l}\} = \{q_{j_1}, \ldots, q_{j_k}\}$. The same type of metrics can be define to compare the fault candidates only.

Definition 17 (Fault Candidate Distance) *The Fault Candidate Distance* $\mathrm{FCD}(\Delta_1, \Delta_2)$ *is*

$$|\{F, \exists (q, F) \in \Delta_1 \setminus \Delta_2\}| + |\{F, \exists (q, F) \in \Delta_2 \setminus \Delta_1\}|.$$

When performing off-line diagnosis, we may not be interested in estimating the current belief state as it is not necessary to provide further diagnosis updates, in this case then, BSD is not the most important metrics to look at, but it is FCD as it is the one that only takes into account the difference between fault candidates.

These previous two metrics compare the diagnoses as a difference of information and has the advantage to be a mathematical distance in order to measure how far from each other the diagnoses are. However, this distance does not compare carefully the common candidates. Hence the introduction of the notion of accuracy and precision.

Definition 18 (Accuracy)

$$\mathrm{Acc}(\Delta_1, \Delta_2) = \frac{|\Delta_1 \cap \Delta_2|}{|\Delta_1 \cup \Delta_2|}$$

$\mathrm{Acc}(\Delta_1, \Delta_2)$ is a ratio that states the number of common candidates in Δ_1 and Δ_2. It tends to 1 when $\Delta_1 = \Delta_2$, it tends to 0 if no common candidate exists. This metrics can especially be utilized when the diagnosis Δ_{real} is known. Indeed, for any diagnosis Δ_1, $\mathrm{Acc}(\Delta_1, \Delta_{real})$ is 0 if Δ_1 does not contain the real candidate. It can also be utilized when the theoretical solution (Definition 6) of $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS})$ is known: any accurate algorithm providing the diagnosis Δ' is necessarily such that $\mathrm{Acc}(\Delta, \Delta') = 1$. Last but not least, by analyzing $\mathrm{Acc}(\Delta, \Delta_{real})$, if it is zero, it also means that the model is not sound and needs to be redesigned [1].

Definition 19 (Precision)

$$\mathrm{Prc}(\Delta_1, \Delta_2) = \min\left(\frac{|\Delta_1|}{|\Delta_2|}, \frac{|\Delta_2|}{|\Delta_1|}\right).$$

The precision checks the ratio between the number of candidates. If $\mathrm{Prc}(\Delta_1, \Delta_2) = 1$, it means that Δ_1 and Δ_2 have the same precision. If the precision is lower than 1, the closer it is to 0, the higher the difference of ambiguity is between Δ_1 and Δ_2. By comparing a diagnosis Δ_1 to the solution Δ of $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS})$, we can measure the precision level of the algorithm that provided Δ_1. Two cases hold:

1. $|\Delta_1| > |\Delta|$ means that the algorithm is not as precise as the accurate algorithm. In this case, it means that the algorithm does not take benefit of the full observability represented in SD. It may be due to the fact the algorithm is computing an approximation of the solution.
2. $|\Delta_1| < |\Delta|$ means that the algorithm only returns partial solutions in the sense that even if the problem is sound, the solution provided by the algorithm may be just wrong.

Precision and accuracy are complementary metrics, as precision does not measure anything about common diagnosis. A less accurate diagnosis may be more precise than a more accurate diagnosis. Ultimately, for a given sound problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS})$ and its solution Δ, the objective is to design an algorithm that provides Δ' such that $\mathrm{BSD}(\Delta', \Delta) \to 0$, $\mathrm{FCD}(\Delta', \Delta) \to 0$, $\mathrm{Acc}(\Delta', \Delta) \to 1$ and $\mathrm{Prc}(\Delta', \Delta) \to 1$.

4.2 Time metrics

Definition 20 (Horizon) *The Horizon* $H_{\mathrm{ALG}}(i)$ *is the time duration required by the on-line algorithm* ALG *to solve* $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS}_i)$.

The Horizon of a perfect diagnosis algorithm is $\forall i, H(i) = 0$. The Horizon measures the capability of an on-line algorithm to *follow* the flow of observations. For example, if $H_{\mathrm{ALG}}(i) > t_{i+1}$, then ALG is not able to provide a final diagnosis for the problem $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS}_i)$ before it has to solve $(\mathrm{SD}, \mathrm{FAULTS}, \mathrm{OBS}_{i+1})$. The consequence is that ALG can have cumulative delays which show that ALG has weak on-line performances.

4.3 Reality metrics

In [7], other metrics are defined. As opposed to the one presented above, they require the knowledge of the real diagnosis. Such metrics measure the quality of the couple $(\mathrm{SD}, \mathrm{ALG})$ as a whole, in the sense that to improve the values measured by these metrics, either SD or ALG or both must be improved. Especially, the improvement of SD usually consists of improving the soundness of the model [1] or the diagnosability of the model and the underlying system by sensor placement [17],[13].

Definition 21 (False Positive) *The number of false positive (*FP*) is the number of faults in* Δ_{ALG} *that are not in* $\Delta_{real} = (q_r, F_r)$*:*

$$\mathrm{FP} = |\{f \in \mathrm{FAULTS}, \exists (q, F) \in \Delta_{\mathrm{ALG}} \wedge f \notin F_r\}|$$

FP usually indicates a lack of diagnosability.

Definition 22 (False Negative) *The number of false negative (*FN*) is the number of faults in* Δ *such that:*

$$\mathrm{FN} = |\{f \in F_r, \forall (q, F) \in \Delta_{\mathrm{ALG}}, f \notin F\}|.$$

FN usually indicates a lack of soundness.

5 FIRST EXPERIMENTS

In this section, we present a set of experiments about two basic algorithms denoted FBFS (Forward Breadth-First Search) and FDFS (Forward Depth-First Search) in order to show the effectiveness of the presented metrics. As stated above, we are not concerned in this section about the quality of the model SD, OBS such as diagnosability and soundness that are out of scope of this paper and requires simulated scenarios and a fault injection simulator to compute for instance the accuracy of the algorithm with respect to the real diagnosis. In these experiments, SD is assumed to be sound (the real trace is represented in the model). The purpose is to compare some algorithms with each other on the same diagnosis problem and to compare their performance.

5.1 Algorithm descriptions

The algorithms to run these first experiments have been chosen for their simple description and not for their efficiency. The aim here is to show how the metrics are able to characterize the behaviour of algorithms. As explained in Section 2.2, the starting point of both algorithms is N automata and their synchronization law which implicitly represent SD, a given set of faults FAULTS to diagnose and the sequence of observations $\text{OBS} = \{(o_1, t_1), \ldots, (o_m, t_m)\}$.

5.1.1 *Forward breadth-first search:* FBFS

The principle of the FBFS algorithm is to start from an initial belief state $\mathcal{I}$[2] and searches with help of a Breadth-First Search (BFS) strategy the set of traces of the global model M that can emit OBS and extracts from them the diagnosis candidates. Note that by construction this algorithm is sound as the BFS guarantees it will find out the complete set of traces. The search space (similar to the behavioural space in [9]) is recursively defined as follows. For each candidate of $(q, F) \in \mathcal{I}$, we associate the state $s = (q, F, 0)$ where 0 means that no observation has occurred. Then, let $s = (q, F, k)$ be a state of the search space then $s' = (q', F', k')$ is a state of the search space iff there exists a transition $q \xrightarrow{e} q'$ in M and one of the following condition holds:

1. e is observable, $F = F'$ and $k' = k + 1$;
2. $e \in$ FAULTS, $F' = F \cup \{e\}$ and $k' = k$;
3. e is neither faulty nor observable and $F = F$ and $k' = k$.

The set of states s' defined as above is denoted $next(s, e)$. Finally FBFS is simply defined as follows by the use of a queue (First In First Out):

```
FBFS(M,FAULTS,OBS,I): output D
do
  for(s=(q,F,0) : (q,F) in I)
    push(queue,s);
    visited(s) <- true;
  endfor

  while(not empty(queue))
    s=(q,F,k) <- pop(queue);
    if(k=m) then insert((q,F),D);
    else
      for(s' : next(s,e), e event of M)
        if ((e = o_k or not_observable(e))
            and not (visited(s')))
        then
          push(queue,s'); visited(s') <- true
        endif
      endfor
    endif
  endwhile
done
```

[2] In Section 2.2, we supposed the initial belief state is $\{(q_0, \emptyset)\}$ however, here, for the sake of generality, we do not make any assumption about it, it can be any set of diagnosis candidates.

5.1.2 *Forward depth-first search:* FDFS

The principle of the FDFS algorithm is similar to the one of FBFS except that the search strategy relies on a Depth-First Search. As for its counterpart, this algorithm is sound. The main difference is the use of a stack (Last In First Out) instead of a queue.

```
FDFS(M,FAULTS,OBS,I): output D
do
  for(s=(q,F,0) : (q,F) in I)
    push(stack,s);
    visited(s) <- true;
  endfor

  while(not empty(stack))
    s=(q,F,k) <- pop(stack);
    if(k=m) then insert((q,F),D);
    else
      for(s' : next(s,e), e event of M)
        if ((e = o_k or not_observable(e))
            and not (visited(s')))
        then
          push(stack,s'); visited(s') <- true
        endif
      endfor
    endif
  endwhile
done
```

5.2 Generation of the Diagnosis Problem

In this paper, as opposed to usual Model-Based Diagnosis papers, we decide to run experiments on a purely random Diagnosis Problem. In order to generate this problem, we develop a random generator for Diagnosis Problem with help of the DIADES tools developed in CNRS-LAAS [8]. The generation of the illustrating model is in three stages.

1. Topology generation: the topology (see Section 2.1) is randomly chosen with help of the tool `generate_topology`. The topology of this example has been generated by fixing the number of components to 10.
2. Component generation: for each component of the topology, the behaviour of this component (see Definition 1) is randomly generated with help of the tool `des_generate` that takes as input the previously generated topology and thus generates a set of 10 automata (see Table 1) and their synchronization model (see Section 2.1). Most importantly, `des_generate` generates automata that are globally consistent in the sense that the global model exists and each local transition of each automaton belongs at least once to the global model. The size of the global model is big enough so that

Diades (with its tool `global_model`) was not able to compute it due to lack of computational resources (memory resources) [3], it has given up after building 185 000 states and 1 000 000 transitions (theoretically, the number of states for this example is lower than 10^{10} which the number of states in the free product of the 10 automata).

3. To finally define FAULTS and OBS, we use the tool `simulate` that takes as input the set of 10 automata and randomly generates a scenario of a fixed observable size. The scenario utilized in the following example consists of a sequence of 10 observations. To define FAULTS, we selected a set of 9 non-observable events as faults.

Comp	StateNb	TransNb	Comp	StateNb	TransNb
$C0$	7	16	$C1$	20	46
$C2$	5	15	$C3$	20	45
$C4$	9	22	$C5$	8	12
$C6$	15	31	$C7$	5	17
$C8$	18	42	$C9$	8	23

Table 1. Size of the randomly generated automata for this example

5.3 First Results

This subsection presents some experiments that illustrate performance metrics introduced in Section 4.1 and the way to use them for comparing the behaviour of diagnosis algorithms to each other. Figures 1, 2, 3, 4 present the behaviour of the metrics of Section 4.1 with the algorithms FBFS and FDFS for the diagnosis problem (SD, FAULTS, OBS) that is described in the previous section. For every Figure, there are three curves. For a given metrics m, the thick curve represents $m(\Delta_{\mathrm{FDFS}}(t), \Delta)$ where Δ is the solution of (SD, FAULTS, OBS) (see Definition 6) and $\Delta_{\mathrm{FDFS}}(t)$ is the partial diagnosis (a set of candidates) that FBFS is able to provide at time t. The thin plain curve represents $m(\Delta_{\mathrm{FBFS}}(t), \Delta)$ and finally the dashed curve represents $m(\Delta_{\mathrm{FDFS}}(t), \Delta_{\mathrm{FBFS}}(t))$.

Figure 1 illustrates the Belief State Distance. At time 0, the distance between the initial diagnosis $(q_0, \varnothing)$ and the solution is in this case 230. At time 50, FDFS starts to produce new candidates that improves the BSD as it is getting closer to the solution. The FDFS curve really shows the overall behaviour of a DFS strategy, when the BSD is stable for a while, it means that FDFS keeps backtracking. As opposed to FDFS, FBFS takes longer to provide the first candidates and the final solution.

This difference can be clearly seen on $\mathrm{BSD}(\Delta_{\mathrm{FDFS}}(t), \Delta_{\mathrm{FBFS}}(t))$ that keeps growing from time 40 to time 80 which shows that FBFS and FDFS are drastically different algorithms. FBFS accelerates the computation of candidates in the end whereas DFS provides candidates with a more stable speed.

Figure 2 illustrates the Fault Candidate Distance. As opposed to BSD,FCD is very small at initial time. It is due to the fact that the initial diagnosis $(q_0, \varnothing)$ has an empty set of fault candidates whereas the solution diagnosis contains 9 fault candidates only (but many possible states associated to them). FDFS provides the correct fault candidates at time 30, whereas FBFS provided them at time 50.

[3] At present, the way the tool `global_model` is implemented is far from being optimal.

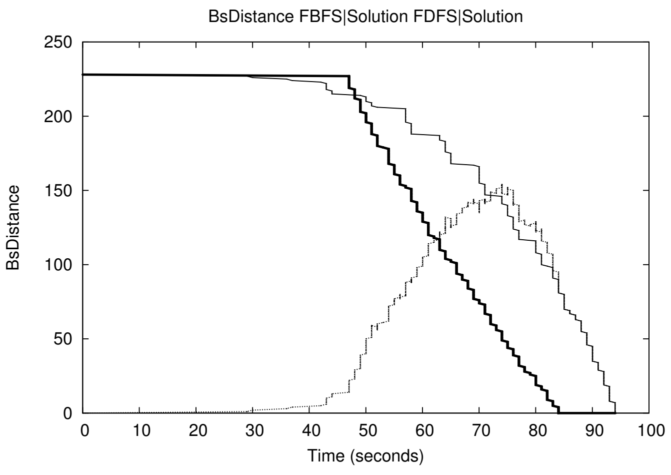

Figure 1. BSD metrics

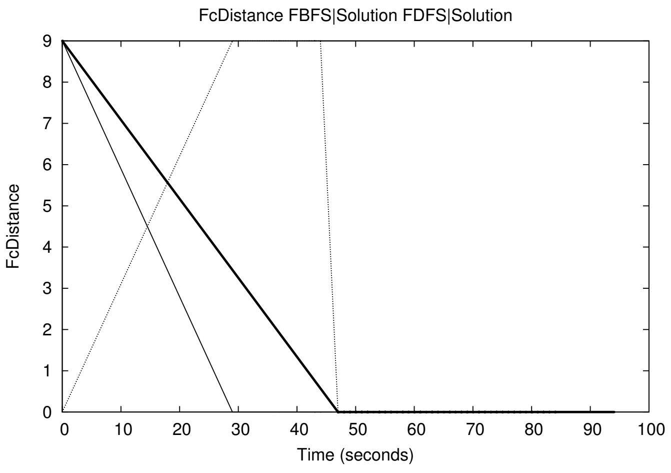

Figure 2. FCD metrics

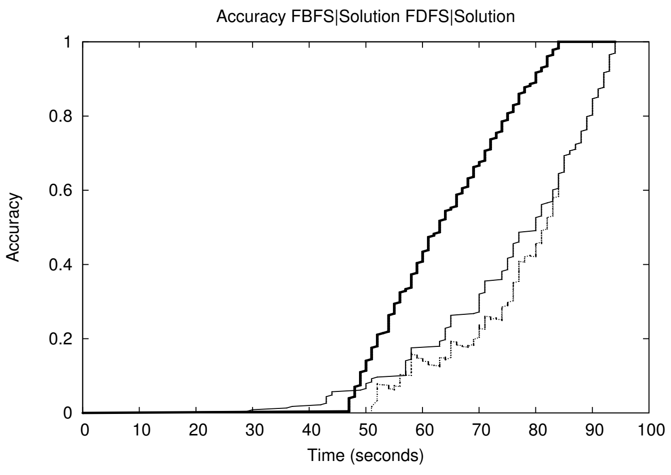

Figure 3. Accuracy metrics

Figures 3 and 4 respectively describe the overall accuracy metrics and precision metrics (embedding belief states and fault candidates). As noticed by analyzing BSD and FCD, the accuracy of FDFS clearly improves earlier than the accuracy of FBFS. Both accuracy tend to 1 which results from the fact that both algorithms are sound on the given problem. Finally, $\mathrm{Acc}(\Delta_{\mathrm{FDFS}}(t), \Delta_{\mathrm{FBFS}}(t))$ also tends to one that shows both algorithms converge on the same solution. As far as the precision is concerned, the behaviour is rather similar to accuracy where FBFS and FDFS are compared to the solution Δ. The most interesting metrics is in this case $\mathrm{Prc}(\Delta_{\mathrm{FDFS}}(t), \Delta_{\mathrm{FBFS}}(t))$ which measures the relative precision of both algorithms over time. Due to the fact that the strategies of FBFS and FDFS are drastically different the behaviour of the relative precision is rather erratic till it converges to 1.

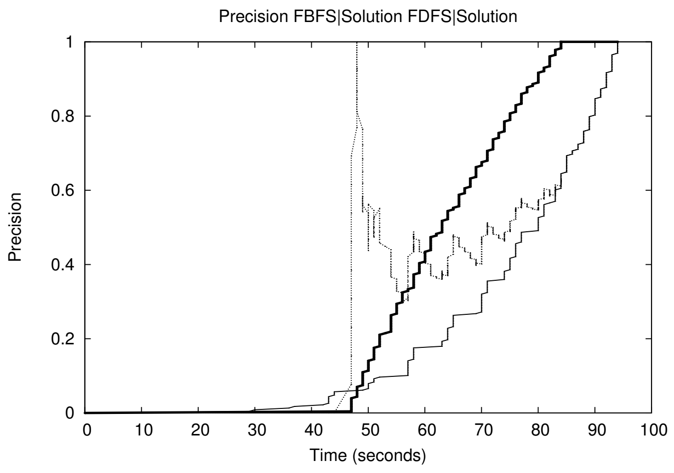

Figure 4. Precision metrics

6 CONCLUSION

In this paper, we discuss the problem of evaluating the performance of diagnosis algorithms that solve the Fault-Diagnosis problem from an experimental point of view. For this special problem, there is a clear distinction between the model part (a set of automata or any equivalent representation) and the algorithm that solves the problem. That is the main reason of introducing a set of proper metrics for algorithms only and for introducing a framework to randomly generate models as it is proposed with the tool DIADES. Of course the experiments presented here are very preliminary and we expect in the very close future to generate a full set of benchmarks and to analyze more sophisticated algorithms with the ones developed in our previous work [12],[16] as a priority. The model generator we are using defines a set of parameters (number of nodes, connectivity, number of states, number of observable events, . . .) that should be interesting to play with in order to generate very different type of DES models and thus sees how algorithms perform on them. There will be two main objectives, that, we actually believe, are essential for the MBD community:

1. to have a clear understanding about how the available diagnosis algorithms scale on the same problems;
2. and more importantly, to have a better understanding about how to improve the efficiency of the algorithms by closer comparative analyses of one another.

REFERENCES

[1] N. Belard, Y. Pencolé, and M. Combacau, 'A theory of meta-diagnosis: Reasoning about diagnostic systems', in *Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 731–737, (2011).

[2] A. Benveniste, E. Fabre, S. Haar, and C. Jard, 'Diagnosis of asynchronous discrete event systems, a net unfolding approach', *IEEE Transactions on Automatic Control*, **48**(5), 714–727, (2003).

[3] L. Console, C. Picardi, and M. Ribaudo, 'Process algebras for systems diagnosis', *Artificial Intelligence*, **142**(1), 19–51, (2002).

[4] R. Debouk, S. Lafortune, and D. Teneketzis, 'A coordinated decentralized protocol for failure diagnosis of discrete event systems', *Journal of Discrete Event Dynamical Systems: Theory and Application*, **10**, 33–86, (2000).

[5] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva, 'Diagnosis of discrete-event systems using satisfiability algorithms', in *American National Conference on Artificial Intelligence (AAAI-07)*, (2007).

[6] Pr. Kan John and Al. Grastien, 'Local consistency and junction tree for diagnosis of discrete-event systems', in *Eighteenth European Conference on Artificial Intelligence (ECAI-08)*, (2008).

[7] T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman, 'Towards a framework for evaluating and comparing diagnosis algorithms', in *20th International Workshop on Principles of Diagnosis (DX-09)*, pp. 373–381, (2009).

[8] Universit de Toulouse LAAS-CNRS. Diades: Diagnosis of discrete-event systems. http://homepages.laas.fr/ypencole/diades.

[9] G. Lamperti and M. Zanella, *Diagnosis of active systems*, Kluwer Academic Publishers, 2003.

[10] G. Lamperti and M. Zanella, 'Diagnosis of discrete-event systems by separation of concerns, knowledge compilation, and reuse', in *16th European Conference on Artificial Intelligence (ECAI-04)*, pp. 838–842, (2004).

[11] G. Lamperti and M. Zanella, 'Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques', *Artificial Intelligence*, **170**(3), 232–297, (2006).

[12] Y. Pencolé and M.-O. Cordier, 'A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks', *Artificial Intelligence*, **164**, 121–170, (May 2005).

[13] P. Ribot, Y. Pencolé, and M. Combacau, 'Design requirements for the diagnosability of distributed discrete event systems', in *19th International Workshop on Principles of Diagnosis (DX'08)*, Blue Mountains, Australia, (September 2008).

[14] J. Rintanen and A. Grastien, 'Diagnosability testing with satisfiability algorithms', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 532–537, (2007).

[15] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, 'Diagnosability of discrete event system', *IEEE Trans. on Automatic Control*, **40**(9), 1555–1575, (1995).

[16] A. Schumann, Y. Pencolé, and S. Thiébaux, 'A spectrum of symbolic on-line diagnosis approaches', in *22nd American National Conference on Artificial Intelligence (AAAI)*, pp. 335–340, (2007).

[17] L. Travé-Massuyès, T. Escobet, and R. Milne, 'Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem', in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI'01*, volume 1, pp. 551–556, (2001).

[18] F. Xue, L. Yan, and D. Zheng, 'Fault diagnosis of distributed discrete event systems using obdd', *Informatica*, **16(3)**, 431–448, (2005).

Computing Manifestations of Max-Size Min-Cardinality Ambiguity Groups

Alexander Feldman and **Johan de Kleer** and **Gregory Provan** [1]

Abstract. The application of Model-Based Diagnosis to systems that are under-observed (e.g., sensor-lean systems) is severely hindered by the ambiguity of the diagnostic result. In the worst-case, even in very restricted frameworks such as the one presented in this paper, an observation may lead to an exponential number of diagnoses. This is the case even if we impose a minimality criterion such as cardinality-minimal diagnoses. To solve this problem researchers have proposed a number of information gathering approaches such as probing and active-testing. There is little literature however, on evaluating the performance of these information-gathering algorithms. In this paper we analyze a new class of observations that maximize the size of the minimal-cardinality (MSMC) ambiguity group. We show a probing framework for which these observation lead to worst-case probing sessions. We exhaustively compute these MSMC initial observations for a benchmark of 74XXX digital circuits.

1 Introduction

Model-Based Diagnosis (MBD) aims to compute, given a model SD and an observation α, diagnoses, minimal under some minimality criterion, e.g., the minimal-cardinality set of faulty components. Since the model SD is known *a priori*, much work has been devoted to optimizing the inference process by pre-processing SD. However, little work has focused on how α affects the inference process. This paper focuses on how α affects the cardinality and minimality properties of diagnoses. We define a metrics, MSMC, which addresses diagnostic ambiguity (or indistinguishability of diagnoses).

Prior work on probing [3] is multi-valued in it does not restrict the variables domains to Boolean values. This makes the proposed approach very useful in practical situations, however it makes the analysis of algorithmic performance more difficult. In this paper we propose a strictly propositional framework that allows more intuitive presentation of assumptions, and analysis of algorithms and properties.

The contributions of this paper are as follows. (1) We define a new class of MSMC observation vectors that are worst-case scenarios for a class of information gathering algorithms and probing in particular. (2) We provide a formal framework for the evaluation of probing algorithms. (3) In the proposed framework, we show a probing algorithm that uses a myopic one-step look-ahead to compute optimal probe variables. We show that MSMC observation vectors are worst-case scenarios for this probing algorithm. (4) We compute MSMC properties for a class of 74XXX benchmarks.

The majority of existing MBD research as well as this paper consider only realistic cases in which the original "injected" faults do not change their location. An interesting alternative to this is a case proposed by de Kleer that we call "The MBD Game". The board of this game is a digital circuit. In the beginning of the game the antagonist "injects" a fault that stays hidden from the protagonist. In each turn of the game, the protagonist proposes a measurement (a probe), the antagonist gives the value of this measurement and changes the location of the fault so the protagonist does not find it. The goal of the protagonist is to minimize the number of probes before finding the fault while the antagonist aims at the opposite (maximizing the number of probes). In a subsequent game the protagonist and antagonist change roles and the winner is the one who uses a smaller number of probes to uniquely determine all faults.

2 Related Work

To the best of our knowledge, we are the first to define the notion of an MSMC observation vector.

Early work [3] aimed at diagnostic convergence by computing a probe sequence for reducing diagnostic entropy using a myopic search strategy. This paper complements this work by providing a strict probing framework in which we can show worst-case scenarios.

Probing is not the only way to perform diagnostic information gathering. Another approach is active testing [8] in active testing one computes a set of optimal control settings that lead to observations of small cardinality-minimal ambiguity groups. The MSMC framework presented in this paper is also a bound on the performance of this class of algorithms.

The material presented in this paper shows an approach to evaluating the performance of the information gathering part of diagnostic algorithms in worst-case scenarios. The international diagnostic competition DXC [4] has similarly evaluated algorithms. The main goal there, however, is the comparison of different diagnostic algorithms and not stress-testing under worst-case conditions. This work may facilitate similar diagnostic competitions in the future by allowing algorithms to compete on difficult observation vectors.

A problem that is related to MSMC is that of computing observations leading to cardinality-minimal diagnoses of maximal cardinality. These observations are called MFMC observations and are studied in [6].

3 Concepts and Definitions

Our discussion continues by formalizing some MBD notions. This article uses the traditional diagnostic definitions [3], except that we use propositional logic terms (conjunctions of literals) instead of sets of failing components.

[1] University College Cork, email: a.feldman@ucc.ie, dekleer@parc.com, g.provan@cs.ucc.ie

Central to MBD, a *model* of an artifact is represented as a Well-Formed Propositional Formula ($\mathbf{Wff}$) over some set of variables. We discern subsets of these variables as *assumable* and *observable*.[2]

Definition 1 (Diagnostic System)**.** A diagnostic system DS is defined as the quadruple DS $= \langle$SD, COMPS, IN, OUT, INT$\rangle$, where SD is a propositional theory over a set of variables V, COMPS $\cup$ IN $\cup$ OUT $\subseteq V$, COMPS is the set of assumables, IN is the set of primary inputs, OUT is the set of primary outputs, and INT $= V \setminus \{$COMPS $\cup$ IN $\cup$ OUT$\cup\}$. The set of observables OBS is defined as OBS $=$ IN $\cup$ OUT.

Throughout this article we restrict SD to propositional theories derived from Boolean circuits. We assume that SD $\not\models \perp$, i.e., SD is not faulty (does not lead to diagnoses) when there is no observation. We also assume that the sets IN, OUT, and COMPS are disjoint. Further, we assume that the model SD is acyclic, testable, and connected, i.e., starting from a primary input in IN we can always reach a primary output in OUT, thus defining direction of each connection (we will illustrate this with an example).

The internal variables of SD are all variables in V that are neither assumables nor primary inputs nor primary outputs, i.e., $V \setminus \{$IN $\cup$ OUT $\cup$ COMPS$\}$.

Let COMPS $= \{h_i\}$ for $i = 1, 2, \ldots, n$. We use positive assignments $h_i = \mathbf{True}$, or simply positive literals h_i, to denote healthy components; conversely, we use negative assignments $h_i = \mathbf{False}$, or negative literals $\neg h_i$, to denote faulty components. Other authors use different mnemonics for this: some denote faulty components with "ab" for abnormal, while others denote healthy components using "ok".

Not all propositional theories used as system descriptions are of interest to MBD. Diagnostic systems can be characterized by a restricted set of models, the restriction making the problem of computing diagnosis amenable to algorithms like the ones presented in this article. We consider two main classes of models.

Definition 2 (Weak-Fault Model)**.** A diagnostic system DS $= \langle \text{SD}, \text{COMPS}, \text{IN}, \text{OUT}, \text{INT} \rangle$, OBS $=$ IN $\cup$ OUT, belongs to the class $\mathbf{WFM}$ iff for COMPS $= \{h_1, h_2, \ldots, h_n\}$, SD is equivalent to $(h_1 \Rightarrow F_1) \wedge (h_2 \Rightarrow F_2) \wedge \ldots \wedge (h_n \Rightarrow F_n)$ and COMPS $\cap V' = \emptyset$, where V' is the set of all variables appearing in the propositional formulae $F_1, F_2, \ldots, F_n$.

Weak-fault models are sometimes referred to as models with *ignorance of abnormal behavior* [2], or *implicit fault systems*. Alternatively, a model may specify the faulty behavior for its components [14]. In the following definition, with the aim of simplifying the formalism throughout this article, we adopt a slightly restrictive representation of faults, allowing only a single fault mode per assumable variable. This can be easily generalized by introducing multi-valued logic or suitable encodings [12, 5].

Definition 3 (Strong-Fault Model)**.** A diagnostic system DS $= \langle \text{SD}, \text{COMPS}, \text{IN}, \text{OUT}, \text{INT} \rangle$, OBS $=$ IN $\cup$ OUT, belongs to the class $\mathbf{SFM}$ iff SD is equivalent to $(h_1 \Rightarrow F_{1,1}) \wedge (\neg h_1 \Rightarrow F_{1,2}) \wedge \cdots \wedge (h_n \Rightarrow F_{n,1}) \wedge (\neg h_n \Rightarrow F_{n,2})$ such that $1 \leq i, j \leq n, k \in \{1, 2\}$, $\{h_i\} \subseteq$ COMPS, $F_{j,k} \in \mathbf{Wff}$, and none of h_i appears in $F_{j,k}$.

Membership testing for the $\mathbf{WFM}$ and $\mathbf{SFM}$ classes can be performed efficiently in many cases, for example, when a model is represented explicitly as in Def. 2 or Def. 3.

[2] In the MBD literature the assumable variables are also referred to as "component", "failure-mode", or "health" variables. Observable variables are also called "measurable", or "control" variables.

3.1 A Running Example

We use the Boolean circuit shown in Fig. 1 to illustrate many notions and algorithms in this article. The subtractor, shown there, consists of seven components: an inverter, two or-gates, two xor-gates, and two and-gates. The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models the normative (healthy) behavior of an inverter, where the variables i, o, and h represent input, output and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2)$ and an or-gate by $h \Rightarrow (o \Leftrightarrow i_1 \vee i_2)$. Finally, an xor-gate is specified as $h \Rightarrow [o \Leftrightarrow \neg (i_1 \Leftrightarrow i_2)]$.

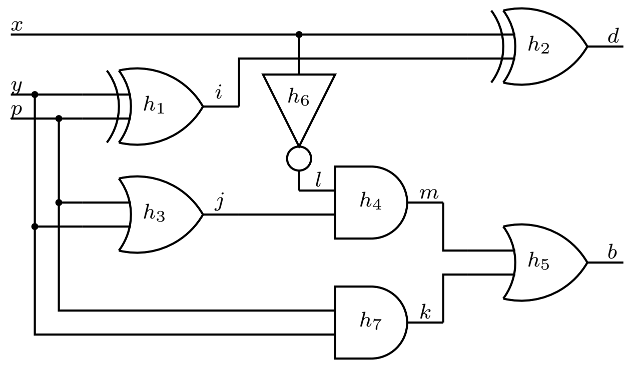

Figure 1. A subtractor circuit

The above propositional formulae are copied for each gate in Fig. 1 and their variables renamed in such a way as to properly connect the circuit and disambiguate the assumables, thus obtaining a propositional formula for the Boolean subtractor, given by:

$$\begin{aligned} \text{SD}_w = &\{h_1 \Rightarrow [i \Leftrightarrow \neg (y \Leftrightarrow p)]\} \wedge \\ &\{h_2 \Rightarrow [d \Leftrightarrow \neg (x \Leftrightarrow i)]\} \wedge \\ &[h_3 \Rightarrow (j \Leftrightarrow y \vee p)] \wedge [h_4 \Rightarrow (m \Leftrightarrow l \wedge j)] \wedge \\ &[h_5 \Rightarrow (b \Leftrightarrow m \vee k)] \wedge [h_6 \Rightarrow (x \Leftrightarrow \neg l)] \wedge \\ &[h_7 \Rightarrow (k \Leftrightarrow y \wedge p)] \end{aligned} \tag{1}$$

A strong-fault model for the Boolean circuit shown in Fig. 1 is constructed by assigning fault-modes to the different gate types. We will assume that, when malfunctioning, the output of an xor-gate has the value of one of its inputs, an or-gate can be stuck-at-one, an and-gate can be stuck-at-zero, and an inverter behaves like a buffer. This gives us the following strong-fault model formula for the Boolean subtractor circuit:

$$\begin{aligned} \text{SD}_s = \text{SD}_w \wedge [\neg h_1 \Rightarrow (i \Leftrightarrow y)] \wedge [\neg h_2 \Rightarrow (d \Leftrightarrow x)] \\ (\neg h_3 \Rightarrow j) \wedge (\neg h_4 \Rightarrow \neg m) \wedge (\neg h_5 \Rightarrow b) \\ [\neg h_6 \Rightarrow (x \Leftrightarrow l)] \wedge (\neg h_7 \Rightarrow \neg k) \end{aligned} \tag{2}$$

For both models (SD$_s$ and SD$_w$), the set of assumable variables is COMPS $= \{h_1, h_2, \ldots, h_7\}$ and the set of observable variables is OBS $= \{x, y, p, d, b\}$, where IN $= \{x, y, p\}$ are the primary inputs and OUT $= \{d, b\}$ are the primary outputs.

Note that each component in SD$_w$ or SD$_s$ has inputs and an output. For example, the inverter which is associated with h_6 has x as its input and its output is l. The and-gate h_4 has two inputs: l and j and one output m.

3.2 Diagnosis and Minimal Diagnosis

The traditional query in MBD computes terms of assumable variables which are explanations for the system description and an observation.

Definition 4 (Health Assignment)**.** Given a system DS $= \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{IN}, \mathrm{OUT}, \mathrm{INT}\rangle$, an assignment ω to all variables in COMPS is defined as a health assignment.

A health assignment ω is a conjunction of propositional literals. In some cases it is convenient to use the set of negative or positive literals in ω. These two sets are denoted as $Lit^-(\omega)$ and $Lit^+(\omega)$, respectively.

In our example, the "all nominal" assignment is $\omega_1 = h_1 \wedge h_2 \wedge \cdots \wedge h_7$. The health assignment $\omega_2 = h_1 \wedge h_2 \wedge h_3 \wedge \neg h_4 \wedge h_5 \wedge h_6 \wedge \neg h_7$ means that the two and-gates from Fig. 1 are malfunctioning.

What follows is a formal definition of consistency-based diagnosis.

Definition 5 (Diagnosis)**.** Given a diagnostic system DS $= \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{IN}, \mathrm{OUT}, \mathrm{INT}\rangle$, OBS $=$ IN $\cup$ OUT, an observation α over some variables in OBS, and a health assignment ω, ω is a diagnosis iff SD $\wedge\, \alpha \wedge \omega \not\models \perp$.

There is a total of 96 possible diagnoses given SD_w and an observation $\alpha_1 = x \wedge y \wedge p \wedge b \wedge \neg d$. Example diagnoses are $\omega_3 = \neg h_1 \wedge h_2 \wedge \cdots \wedge h_7$ and $\omega_4 = h_1 \wedge \neg h_2 \wedge h_3 \wedge \cdots \wedge h_7$. Trivially, given a weak-fault model, the "all faulty" health assignment (in our example $\omega_a = \neg h_1 \wedge \cdots \wedge \neg h_7$) is a diagnosis for any instantiation of the observable variables in OBS (see Def. 2).

In the MBD literature, a range of types of "preferred" diagnosis has been proposed. This turns the MBD problem into an optimization problem. In the following definition we consider the common subset-ordering.

Definition 6 (Minimal Diagnosis)**.** A diagnosis $\omega^\subseteq$ is defined as minimal, if no diagnosis $\tilde{\omega}^\subseteq$ exists such that $Lit^-(\tilde{\omega}^\subseteq) \subset Lit^-(\omega^\subseteq)$.

Consider the weak-fault model SD_w of the circuit shown in Fig. 1 and an observation $\alpha_2 = \neg x \wedge y \wedge p \wedge \neg b \wedge d$. In this example, two of the minimal diagnoses are $\omega_5^\subseteq = \neg h_1 \wedge h_2 \wedge h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$ and $\omega_6^\subseteq = \neg h_1 \wedge h_2 \wedge \cdots \wedge h_5 \wedge \neg h_6 \wedge \neg h_7$. The diagnosis $\omega_7 = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$ is non-minimal as the negative literals in $\omega_5^\subseteq$ form a subset of the negative literals in ω_7.

Definition 7 (Subset-Minimal Ambiguity Group)**.** The subset-minimal ambiguity group of a system description SD and an observation α, denoted as $\Omega^\subseteq(\mathrm{SD} \wedge \alpha)$, is defined as the set of all minimal diagnoses of SD $\wedge\, \alpha$.

Note that the set of all minimal diagnoses characterizes all diagnoses for a weak-fault model, but that does not hold in general for strong-fault models [2]. In the latter case, faulty components may "exonerate" each other, resulting in a health assignment containing a proper superset of the negative literals of another diagnosis not to be a diagnosis. In our example, given SD_s and $\alpha_3 = \neg x \wedge \neg y \wedge \neg p \wedge b \wedge \neg d$, it follows that $\omega_8^\subseteq = h_1 \wedge h_2 \wedge \neg h_3 \wedge h_4 \wedge \cdots \wedge h_7$ is a diagnosis, but $\omega_9 = h_1 \wedge h_2 \wedge \neg h_3 \wedge \neg h_4 \wedge \cdots \wedge h_7$ is not a diagnosis, despite the fact that the negative literals in ω_9 ($\{\neg h_3, \neg h_4\}$) form a superset of the negative literals in $\omega_8^\subseteq$ ($\{\neg h_3\}$).

Definition 8 (Number of Minimal Diagnoses)**.** Given a system description SD and an observation α, the number of minimal diagnoses, denoted as $|\Omega^\subseteq(\mathrm{SD} \wedge \alpha)|$, is defined as the size of the subset-minimal ambiguity group $\Omega^\subseteq(\mathrm{SD} \wedge \alpha)$.

Continuing our running example, $|\Omega^\subseteq(\mathrm{SD}_w \wedge \alpha_2)| = 8$ and $|\Omega^\subseteq(\mathrm{SD}_s \wedge \alpha_3)| = 2$. The number of non-minimal diagnoses of $\mathrm{SD}_w \wedge \alpha_2$ is 61.

Definition 9 (Cardinality of a Diagnosis)**.** The cardinality of a diagnosis ω, denoted as $|\omega|$, is defined as the number of negative literals in ω.

Diagnosis cardinality gives us another partial ordering: a diagnosis is defined as *minimal cardinality* iff it minimizes the number of negative literals.

Definition 10 (Minimal-Cardinality Diagnosis)**.** A diagnosis $\omega^\leq$ is defined as minimal-cardinality if no diagnosis $\tilde{\omega}^\leq$ exists such that $|\tilde{\omega}^\leq| < |\omega^\leq|$.

The cardinality of a minimal-cardinality diagnosis computed from a system description SD and an observation α is denoted as $MinCard(\mathrm{SD} \wedge \alpha)$. For our example model SD_w and an observation $\alpha_4 = x \wedge y \wedge p \wedge \neg b \wedge \neg d$, it follows that $MinCard(\mathrm{SD}_w \wedge \alpha_4) = 2$. Note that in this case all minimal diagnoses are also minimal-cardinality diagnoses.

A minimal cardinality diagnosis is a minimal diagnosis, but the opposite does not hold. There are minimal diagnoses that are not minimal-cardinality diagnoses. Consider the example SD_w and α_2 given earlier in this section, and the two resulting minimal diagnoses $\omega_5^\subseteq$ and $\omega_6^\subseteq$. From these two, only $\omega_5^\subseteq$ is a minimal-cardinality diagnosis.

Definition 11 (Minimal-Cardinality Ambiguity Group)**.** The minimal-cardinality ambiguity group of a system description SD and an observation α, denoted as $\Omega^\leq(\mathrm{SD} \wedge \alpha)$, is defined as the set of all minimal-cardinality diagnoses of SD $\wedge\, \alpha$.

Counting the number of diagnoses in $\Omega^\leq(\mathrm{SD} \wedge \alpha)$ gives us the final definition for this section.

Definition 12 (Number of Minimal-Cardinality Diagnoses)**.** The number of minimal-cardinality diagnoses, denoted as $|\Omega^\leq(\mathrm{SD} \wedge \alpha)|$, is defined as the cardinality of $\Omega^\leq(\mathrm{SD} \wedge \alpha)$.

Computing the number of minimal-cardinality diagnoses for the running example results in $|\Omega^\leq(\mathrm{SD}_w \wedge \alpha_2)| = 2$, $|\Omega^\leq(\mathrm{SD}_s \wedge \alpha_3)| = 2$, and $|\Omega^\leq(\mathrm{SD}_w \wedge \alpha_4)| = 4$.

4 Observation Vector Optimization Problems

Consider the set of diagnoses in a subset-minimal ambiguity group $\Omega^\subseteq(\mathrm{SD} \wedge \alpha)$. We can construct a distribution of the subset-minimal diagnoses in $\Omega^\subseteq(\mathrm{SD} \wedge \alpha)$ by counting the number of diagnoses with cardinality $0, 1, 2, \ldots$ and computing how frequently each cardinality appears in $\Omega^\subseteq(\mathrm{SD} \wedge \alpha)$. The distribution of the diagnosis cardinalities in $\Omega^\subseteq(\mathrm{SD} \wedge \alpha)$ is denoted as $\widetilde{\Omega}^\subseteq(\mathrm{SD} \wedge \alpha)$. Note that $\widetilde{\Omega}^\subseteq(\mathrm{SD} \wedge \alpha)$ can be arbitrary, i.e., we can construct a system description SD and an observation α resulting in any $\widetilde{\Omega}^\subseteq(\mathrm{SD} \wedge \alpha)$. In this paper, SD is fixed, and the main focus of our work is how $\widetilde{\Omega}^\subseteq(\mathrm{SD} \wedge \alpha)$ changes for various instantiations of the observation α. In particular we are interested in computing observations α that optimize certain parameters defined on the distribution $\widetilde{\Omega}^\subseteq(\mathrm{SD} \wedge \alpha)$.

Figure 2 shows $\widetilde{\Omega}^\subseteq(\mathrm{SD} \wedge \alpha)$ for a weak-fault model of the 74182 combinatorial circuit (part of the 74XXX/ISCAS85 benchmark, see

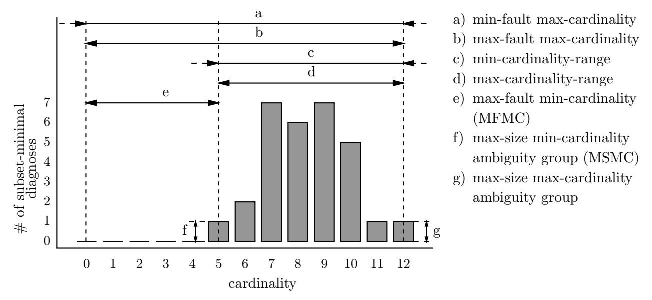

Figure 2. An example distribution of the cardinalities of the subset-minimal ambiguity group for a given observation α. Other observations lead to different distributions. All problems are defined as computing an observation vector (showing all possible observation vectors for this example would add another dimension to the figure) that optimizes certain properties of this distribution. These properties are indicated by arrows.

Sec. 6) and an arbitrary observation α. In addition to that, Fig. 2 illustrates a number of observation vector optimization problems.

From the seven observation vector optimization problems shown in Fig. 2, two are of practical significance to MBD: MFMC and MSMC. We next formally define those.

Problem 1 (MFMC Observation)**.** Given a system DS $= \langle$SD, COMPS, IN, OUT, INT$\rangle$, compute an observation α (defined as Max-Fault Min-Cardinality (MFMC) observation) such that ω is a minimal-cardinality diagnosis of SD $\wedge\ \alpha$ and $|\omega|$ is maximized.

In addition to an MFMC observation, we also refer to an MFMC diagnosis of a model SD. This refers to any of the minimal-cardinality diagnoses $\omega^{\leq}$ of SD $\wedge\ \alpha$ where α is an MFMC observation. The cardinality of this diagnosis is denoted as $\mathit{MFMC}(\text{SD})$ and, next to the associated MFMC observations, this is a key model property we seek to compute.

Problem 2 (MSMC Observation)**.** Given a system DS $= \langle$SD, COMPS, IN, OUT, INT$\rangle$, compute an observation α (defined as Max-Size Min-Cardinality ambiguity group (MSMC) observation) such that $|\Omega^{\leq}(\text{SD} \wedge \alpha)|$ is maximized.

We denote $|\Omega^{\leq}(\text{SD} \wedge \alpha)|$ where α is an MSMC observation as $\mathit{MSMC}(\text{SD})$.

Fig. 2 also illustrates some MBD problems that are less often encountered in practice. The min-fault max-cardinality problem, for example, is to compute the following observation. First, consider the subset-minimal ambiguity group of each different observation (there are $2^{|\text{OBS}|}$ different observations). Second, take the observation that minimizes the number of faults in the maximum-cardinality diagnosis in each subset-minimal ambiguity group.

A related problem that is not illustrated in Fig. 2 is the max-size subset-minimal ambiguity group. The problem is to compute an observation α that maximizes the size of the subset-minimal ambiguity group.

5 Probing

Probing aims to minimize the expected number of diagnoses that result from the possible set of outputs that may occur from the measurement of a given internal (probe) variable.

5.1 Computing the Expected Number of MC Diagnoses

We will compute the expected number of diagnoses for a set of observable variables M ($M \subseteq$ OBS). The initial observation α and the set of MC diagnoses $D = \Omega^{\leq}(\text{SD}, \alpha)$ modify the probability density function of subsequent outputs (observations), i.e., a subsequent observation α' changes its likelihood. The (non-normalized) a posteriori probability of an observation α', given a function $\Omega^{\leq}$ that computes the set of MC diagnoses and an initial observation α, is:

$$\Pr(\alpha'|\text{SD}, \alpha) = \frac{|\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \alpha')|}{|\Omega^{\leq}(\text{SD}, \alpha)|} \tag{3}$$

The above formula computes the probability of a given a priori set of diagnoses restricting the possible outputs, i.e., we assume that the probability is the ratio of the number of remaining diagnoses to the number of initial diagnoses. In practice, there are many α for which $\Pr(\alpha'|\text{SD}, \alpha) = 0$, because a certain fault heavily restricts the possible outputs of a system (i.e., the set of the remaining diagnoses in the numerator is empty).

The expected number of remaining MC diagnoses for a variable set M, given an initial observation α, is then the weighted average of the intersection sizes of all possible instantiations over the variables in M (the weight is the probability of an output):

$$E^{\leq}(\text{SD}, M|\alpha) = \frac{\sum\limits_{\alpha' \in M^*} |\Omega^{\cap}(D, \alpha')| \cdot \Pr(\alpha'|\text{SD}, \alpha)}{\sum\limits_{\alpha' \in M^*} \Pr(\alpha'|\text{SD}, \alpha)} \tag{4}$$

where $D = \Omega^{\leq}(\text{SD}, \alpha)$ and M^* is the set of all possible assignments to the variables in M. Replacing (3) in (4) and simplifying gives us the following definition:

Definition 13 (Expected Minimal-Cardinality Diagnoses Intersection Size)**.** Given a system ATS and an initial observation α, the expected remaining number of MC diagnoses $E^{\leq}(\text{SD}, \text{OBS}|\alpha)$ is defined as:

$$E^{\leq}(\text{SD}, \text{OBS}|\alpha) = \frac{\sum\limits_{\alpha' \in \text{OBS}^*} |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \alpha')|^2}{\sum\limits_{\alpha' \in \text{OBS}^*} |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \alpha')|} \tag{5}$$

where OBS* is the set of all possible assignments to all variables in OBS.

The expected number of remaining MC diagnoses for one variable simplifies the expression in (5) to:

$$E^{\leq}(\text{SD}, v|\alpha) = \frac{p(\text{SD}, v, \alpha)^2 + q(\text{SD}, v, \alpha)^2}{p(\text{SD}, v, \alpha) + q(\text{SD}, v, \alpha)} \tag{6}$$

where

$$p(\text{SD}, v, \alpha) = |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), v)| \tag{7}$$

and

$$q(\text{SD}, v, \alpha) = |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \neg v)| \tag{8}$$

Algorithm 1: Probing framework

```
Input: DS, a diagnostic system,
       DS = ⟨SD, IN, OUT, COMPS, INT⟩
Result: $p$, number of probes, $R \in \mathbb{Z}$
Local variables: $\alpha$, observation term
                 $\omega^{\leq}$, cardinality-minimal diagnosis
                 $z$, probe variable
                 $l$, literal
1 $\langle \alpha, \omega^{\leq} \rangle \leftarrow$ InjectFault(DS)
2 $p \leftarrow 0$
3 while $|\Omega^{\leq}(\mathrm{SD}, \alpha)| \neq 1$ do
4     $z \leftarrow$ ComputeProbe(SD, $\alpha$, INT)
5     $l \leftarrow$ EvaluateProbe(SD, $\alpha$, $\omega^{\leq}$, $z$)
6     $\alpha \leftarrow \alpha \wedge l$
7     INT $\leftarrow$ INT $\setminus z$
8     $p \leftarrow p + 1$
9 return $p$
```

5.2 Probing Algorithm

Algorithm 1 shows a generalized procedure for the evaluation of the performance of probing algorithms. It can be generalized to evaluate the performance of any information gathering procedures (such as active testing [7]), to include probing costs, etc.

Algorithm 1 starts by generating an observation α that leads to a cardinality-minimal diagnosis $\omega^{\leq}$. This is done by a call to the InjectFault subroutine in line 1. Algorithm 1 needs a diagnostic engine that can count the number of cardinality-minimal diagnoses (line 3). The probing algorithm is called in line 4. The probing algorithm returns a variable (probe) that will be "measured". The "measured" values of probe z is computed by the EvaluateProbe auxiliary subroutine in line 5. Methods such as Binary Constraint Propagation (BCP) [10] or SAT solvers are suitable for calculating the value of z given the observation and the injected cardinality-minimal fault. Algorithm 1 evaluates the performance of probing algorithms in terms of the number of probes p.

The following assumptions are made when designing Alg. 1:

Monotone $|\Omega^{\leq}(\mathrm{SD}, \alpha)|$: We restrict ourselves to such system descriptions SD such that if α and β are two observations such that $\alpha \supseteq \beta$ then it holds that $|\Omega^{\leq}(\mathrm{SD}, \alpha)| \geq |\Omega^{\leq}(\mathrm{SD}, \alpha)|$. We can proof that this holds for "well-formed" system descriptions and weak-fault models. The idea is to construct a system of Boolean equations B in the following manner. First, the propositional **Wff** in SD is converted to a Boolean equation in a straightforward manner and the latter is added to B. Second, for each literal $l_i \in \alpha$, an equation of the form $l_i = 1$ or $l_i = 0$ (depending on the polarity of l_i) is appended to B. A system of Boolean equations B' is constructed from SD and β in an analogous way. The solutions of B and B' are the implicants of SD $\wedge\, \alpha$ and SD $\wedge\, \beta$, respectively. Observe, that, due to the fact that $\alpha \supseteq \beta$, the equations in B' are a superset of these in B and both are over the same set of variables. But $S(B') \leq S(B)$, where $S(X)$ denotes the number of solutions in a system X. The above holds also when the solutions of B and B' are ordered according to their cardinality. Hence, if a diagnosis with a cardinality smaller than the smallest cardinality diagnosis in B' exists, it is in B.

Non-ambiguous fault: Given a diagnostic system DS = ⟨SD, IN, OUT, COMPS, INT⟩ we assume that there exists an observation α and an instantiation over a set of variables $P \subseteq$ INT such that $|\Omega^{\leq}(\mathrm{SD}, \alpha)| = 1$. This is easily achievable if SD $\in$ **WFM** and if INT $= V \setminus \{\mathrm{IN} \cup \mathrm{OUT} \cup \mathrm{COMPS}\}$.

No "don't cares" and well-formed SD: We require all SD to be models of well-formed digital circuits. A well-formed digital circuits is constructed from standard AND, OR, NAND, or NOR gates of two or more inputs, from XOR gates, buffers, and inverters. There are no "hanging" wires, each output is connected to the input of another gate or two a primary output. A well-formed circuit does not use any feedback.

Algorithm 2 shows a simple greedy approach to compute the optimal probe variable based on the expected cardinality-minimal intersection size.

The computational performance of Alg. 2 is dominated by the complexity of the diagnostic engine that counts the remaining number of cardinality minimal diagnoses in lines 2 and 3. Assuming that this number decreases monotonically improves the complexity significantly.

Algorithm 2: Probing algorithm

```
Input: SD, a system description
Input: $\alpha$, an observation
Input: INT, a set of probe variables
Result: an optimal probe variable $z \in$ INT
Local variables: $p, q$, number of diagnoses
                 $E, E^{\star}$, reals, expected number of diagnoses
                 $v$, candidate probe variable
1 foreach $v \in$ INT do
2     $p \leftarrow |\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha), v)|$
3     $q \leftarrow |\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha), \neg v)|$
4     $E = \frac{p^2 + q^2}{p + q}$
5     if $E^{\star} < E$ then
6         $E^{\star} \leftarrow E$
7         $z \leftarrow v$
8 return $z$
```

We can see that the number of probes k required for the uniquely (non-ambiguous) isolation of a fault can be an arbitrary value $0 \leq k \leq |\mathrm{INT}|$. There are circuits such as n chained buffers (or inverters) for which Alg. 2 can isolate a single-fault in $k = \log n$ calls. In the worst-case, Alg. 2 needs to probe each probe variable.

We can see that the performance of Alg. 2 is determined by SD and the injected fault $\omega^{\leq}$ injected by Alg. 1. As SD is given, the only variable that Alg. 1 can modify is the initial set of cardinality-minimal diagnoses. It is straightforward to show then that a worst-case scenario for Alg. 2 is when InjectFault(DS) returns an MSMC fault.

6 Experimental Results

This section discusses some results from an implementation of the algorithms described in the previous sections.

6.1 Experimental Setup, Simplification Results and Bounds

We have experimented on the medium-sized circuits from the 74XXX family [11]. Table 1 provides a summary of the 74XXX circuits. The number of inputs, outputs and components are given in the third, fourth, and fifth column of Table 1, respectively.

Table 1. 74XXX circuits

Name	Description	$\lvert\text{IN}\rvert$	$\lvert\text{OUT}\rvert$	$\lvert\text{COMPS}\rvert$
74182	4-bit CLA	9	5	19
74L85	4-bit comparator	11	3	33
74283	4-bit adder	9	5	36
74181	4-bit ALU	14	8	65
c6288	32-bit multiplier	32	32	2 416

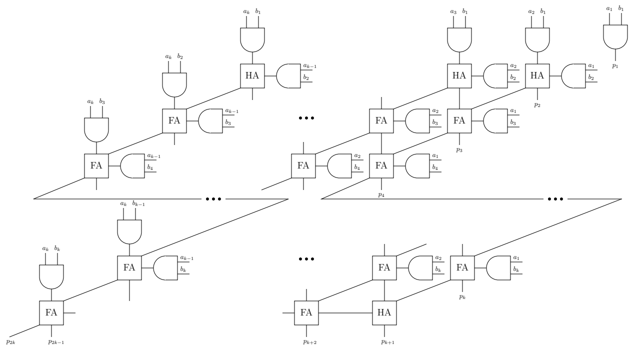

Figure 3. n-bit parallel multiplier ($n = 2k$). For c6288, $n = 32$.

In addition to the 74XXX circuits we have also considered a variation of the c6288 multiplier, part of the ISCAS85 benchmark. Despite the large number of components, c6288 has very regular structure: it is composed entirely of Boolean adders and and-gates as shown in Fig. 3 (the high-level structure of c6288 and Fig. 3 are due to the reverse-engineering efforts of [11]). Inspecting the reverse-engineered c6288 allowed us to construct similar smaller multipliers that have between 1 and 32 outputs. The smallest of them is amenable to an exhaustive approach. The regular structure of c6288 allows us to analytically hypothesize about the MFMC/MSMC properties of c6288 and the whole family of multipliers that have the same high-level structure. For example, one can show experimentally that for an n-bit multiplier having the structure of Fig. 3, it always holds that $\mathit{MFMC}^{\leq} = n$.

6.2 Solving the 74XXX Models Exhaustively

We first tried to exhaustively enumerate the space of all input/output assignments. For 74182, 74L85, and 74283 the size of this space is 16 384 (14 observable variables), while for 74181, it is 4 194 304 (22 observable variables). We used two state-of-the-art complete diagnostic solvers: HA* [9] and NGDE [1].

By using HA* in combination with cones [13] we computed all minimal-cardinality ambiguity groups for the 74XXX models. 74182 was the only circuit for which we could compute all subset-minimal ambiguity groups (these are different from the cardinality-minimal ambiguity groups). We recomputed all diagnoses with NGDE, which is a completely independent implementation by one of the authors of this paper, and the HA* and NGDE results match. Furthermore, NGDE did not use cones for 74XXX, while HA* did, thus independently verifying the correctness of the MFMC/MSMC values, and the correct implementation of the algorithms for computing minimal diagnoses.

The exhaustive search results of the small circuits are shown in Table 2. We can see that for 74182, 74L85, and 74283, $\mathit{MFMC}(\text{SD}) = \lvert\text{OUT}\rvert$, and for 74181 the MFMC value is smaller than the number of outputs $\lvert\text{OUT}\rvert$. The MSMC value for the 74XXX models grows quickly with increasing model size ($\lvert\text{COMPS}\rvert$).

Table 2. Properties of 74XXX subset-minimal diagnoses

Optimization Problem	74182	74L85	74283	74181
min-fault max-cardinality	1	1	–	–
max-fault max-cardinality	14	10	–	–
min-cardinality-range	0	0	–	–
max-cardinality-range	9	8	–	–
MFMC	5	3	5	7
MSMC	400	468	9 132	42 112

Figure 4 is a two-dimensional histogram of the minimal-cardinality ambiguity groups of the 74XXX models. Figure 4 plots on the z-axis the number of observation vectors leading to a minimal-cardinality ambiguity group of size $\lvert\Omega^{\leq}(\text{SD} \wedge \alpha)\rvert$ (y-axis) and minimal cardinality $\lvert\omega\rvert$ (x-axis). We can see that there are no observations leading to low minimal-cardinality and high ambiguity group size and vice versa. We can also see that, in general, an increase in MFMC leads to an increase in MSMC. Furthermore, MFMC/MSMC observation vectors are relatively rare and the MSMC observation vectors are not always MFMC observation vectors (consider, for example, the histogram of 74181 in Figure 4) and vice-versa.

There are 36 MSMC observation vectors for 74182, for example. Of those, 18 observations lead to a minimal-cardinality diagnosis of cardinality 4 and 18 observations lead to a minimal-cardinality diagnosis of cardinality 5. All MSMC observations lead to nearly MFMC diagnoses. As it is visible from Fig. 4, MFMC observations lead to multiple values for the sizes of the minimal-cardinality ambiguity groups. In 74182, for example, there are 7 MFMC observations that lead to a unique minimal-cardinality diagnosis.

Given a system DS, we denote as $g(\text{DS})$ the probability density function of the minimal-cardinalities of the diagnoses of all observations in DS. Figure 5 shows a histogram of the true minimal-diagnosis cardinalities for the four 74XXX circuits for which we have exhaustively determined $g(\text{DS})$, fitted by a normal distribution.

Figure 5 shows the number of observations per minimal-cardinality. We noticed that a normal distribution fits the empirical data well in Fig. 5 (the standard error for 74182, 74L85, 74283, and 74283 is 154, 244, 100, and 18 955, respectively). This is explained as follows. Given an observation α leading to a k-fault minimal diagnosis, we associate a nominal-diagnosis observation α_n, which may differ from α only in the OUT sub-vector. The number of OUT-values in which α and α_n differ is called the *distance* of α, $D(\text{SD}, \alpha)$. If $n = \lvert\text{OUT}\rvert$ is the number of output variables in SD, then starting from any nominal observation α_n, there are ${}_nC_k$ ways to select a distance-k vector α, each of which corresponds to a diagnosis. In the case where each such diagnosis is a minimum cardinality diagnosis, $g(\text{SD})$ is binomially-distributed.

To understand better why the distribution of the minimal-cardi-

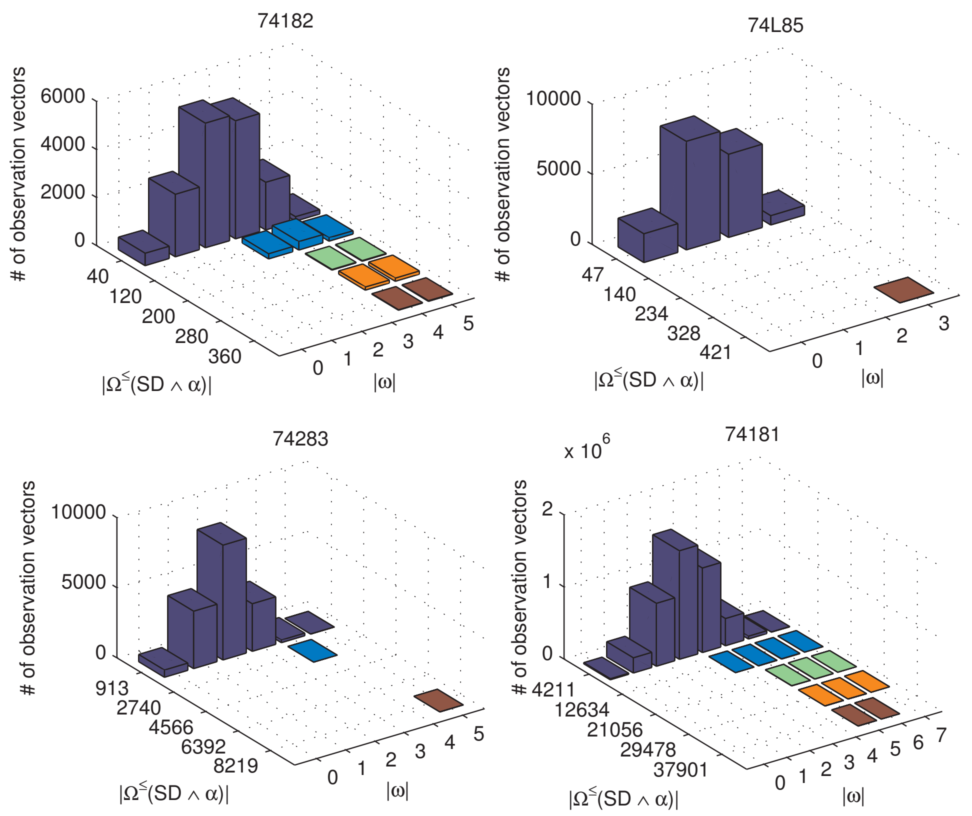

Figure 4. Number of observation vectors vs. cardinality and number of minimal-cardinality diagnoses bivariate histograms for 74XXX

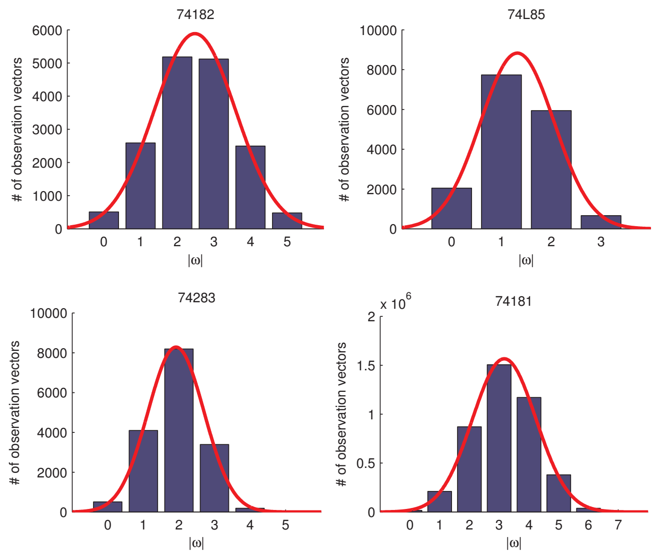

Figure 5. 74XXX minimal-cardinalities distribution

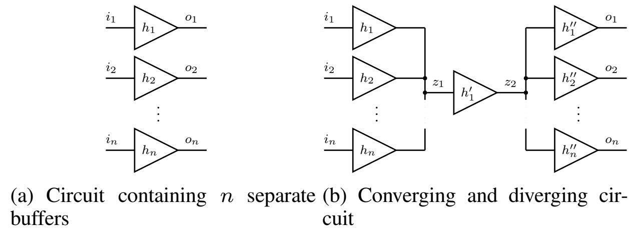
(a) Circuit containing n separate buffers
(b) Converging and diverging circuit

Figure 6. A model with a binomial minimal-cardinality distribution (left) and a model with one nominal minimal-cardinality diagnosis and one single-fault minimal-cardinality diagnosis (right)

nality diagnoses of many circuits can be approximated with a binomial distribution, consider **WFM** of the two synthetic circuits shown in Fig. 6. Both circuits consists of buffers only, where each buffer is modeled as $h \Rightarrow (o \Leftrightarrow i)$. Both circuits have the same input and output variables (IN $= \{i_1, i_2, \ldots, i_n\}$, and OUT $= \{o_1, o_2, \ldots, o_n\}$). The distributions of the minimal-cardinality diagnoses, however, are very different. The model of the circuit shown in Fig. 6(a) has one nominal behavior (health assignment in which all health literals are positive) and n single faults (health assignments in which there is exactly one negative literal). The same circuit has $\frac{n(n-1)}{2}$ double faults, $\frac{n(n-1)(n-2)}{6}$ triple faults, etc. Any observation for the model of the circuit shown in Fig. 6(b), however, leads either to nominal behavior or to the single fault $\neg h_1'$. As we have seen in Fig. 5, the distributions of the 74XXX circuits resemble more the distribution associated with Fig. 6(a). The density mass of all distributions shown in Fig. 5 are skewed to the left and the amount with which a distribution is skewed to the left depends on the masking phenomenon demonstrated in Fig. 6(b).

Although the above model is an approximation, it provides useful bounds on MFMC errors. Let m be the number of bits in the output assignment that differ from the nominal output value. For the 74XXX and ISCAS85 benchmarks, the fraction of "m-flips" resulting in minimal-cardinality diagnoses of cardinality smaller than m is relatively small and does not vary significantly for different m.

Figure 7 shows a histogram of the minimal-cardinality ambiguity group sizes for all 74XXX circuits. We can see that when increasing the minimal-cardinality ambiguity group size, the number of observation vectors decreases rapidly. This depends on the model topology, and is less prevalent in 74182. The MSMC value of 74181, for example, is $42\,112$, and as is visible from Fig. 7, there are relatively few observations leading to such large ambiguity groups.

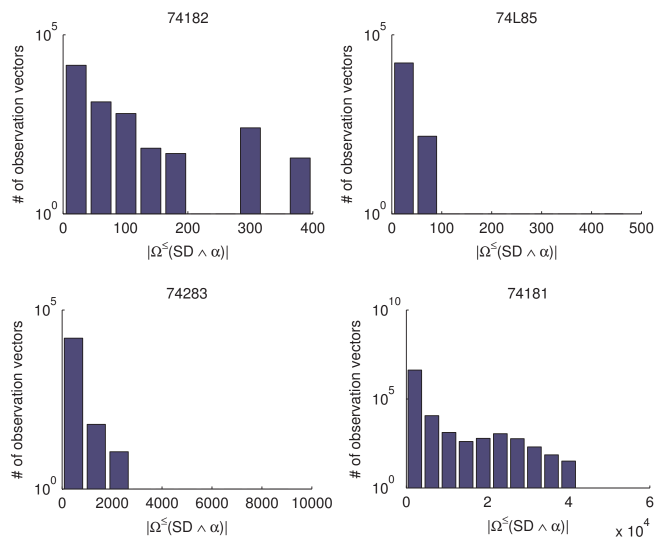

Figure 7. 74XXX minimal-cardinality ambiguity group sizes distribution

Table 3 shows the MFMC and MSMC values of several small multipliers (see Fig. 3). We have created two types of fault-models: in Type I models we have assigned only one health variable to each half-adder or full-adder (i.e., all the gates in an adder fail simultaneously), and in Type II fault-models we have associated an assumable with each logic gate (as everywhere else in this paper). The 2-bit multiplier consists only of a single and-gate, hence all MFMC and MSMC values are trivially 1. For Type I fault-models we can see that the MFMC value of an n-bit multiplier is $n/2+1$ ($n = 2k$, $k \in \mathbb{N}^+$). For Type II multipliers the MFMC value of an n-bit multiplier is n.

Table 3. MFMC and MSMC values of small multipliers

	Type I		Type II	
bits ($\lvert\text{OUT}\rvert$)	MFMC	MSMC	MFMC	MSMC
2	1	1	1	1
4	3	6	4	9
6	4	58	6	3 969
8	5	845	–	–

7 Conclusions

This paper has defined a class of MSMC observation vectors which are the worst-case for the fault ambiguity (or indistinguishability). The MSMC of real-world systems is an important property quantifying the diagnosability of a model, as it shows the maximum number of cardinality-minimal diagnoses that can be returned by observing a set of variables.

We have shown a probing algorithm for which an MSMC observation vector results in the largest number of steps for reducing the initial set of cardinality-minimal diagnoses to a single candidate.

Computing MSMC-related properties of models of real-world artifacts is important for (1) assessing the performance of MBD and information gathering algorithms and (2) better understanding the diagnosability properties of the design.

Computing MSMC is a difficult counting problem and its complexity is hypothesized to be at least the complexity of counting the number of cardinality-minimal diagnoses entailed by a system description and an observation. As a result algorithms that can compute MSMC must utilize properties of the model, such as structure and hierarchy, in order to provide results for systems of practical size.

We have computed MSMC values for the 74XXX models. As a future work we plan to design more efficient MSMC algorithms and to apply them to a class of larger benchmarks.

REFERENCES

[1] Johan de Kleer, 'Minimum cardinality candidate generation', in *Proc. DX'09*, pp. 397–402, (2009).
[2] Johan de Kleer, Alan Mackworth, and Raymond Reiter, 'Characterizing diagnoses and systems', *Artificial Intelligence*, **56**(2-3), 197–222, (1992).
[3] Johan de Kleer and Brian Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**(1), 97–130, (1987).
[4] Alexander Feldman, Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Johan de Kleer, Lukas Kuhn, and Arjan van Gemund, 'Empirical evaluation of diagnostic algorithm performance using a generic framework', *International Journal of Prognostics and Health Management*, 1–28, (2010).
[5] Alexander Feldman, Jurryt Pietersma, and Arjan van Gemund, 'A multi-valued SAT-based algorithm for faster model-based diagnosis', in *Proc. DX'06*, (2006).
[6] Alexander Feldman, Gregory Provan, and Arjan van Gemund, 'Computing minimal diagnoses by greedy stochastic search', in *Proc. AAAI'08*, pp. 919–924, (2008).
[7] Alexander Feldman, Gregory Provan, and Arjan van Gemund, 'FRACTAL: Efficient fault isolation using active testing', in *Proc. IJCAI'09*, (2009).
[8] Alexander Feldman, Gregory Provan, and Arjan van Gemund, 'Stochastic algorithms for sequential model-based diagnosis', *Journal of Artificial Intelligence Research*, **39**, 301–334, (2010).
[9] Alexander Feldman and Arjan van Gemund, 'A two-step hierarchical algorithm for model-based diagnosis', in *Proc. AAAI'06*, (2006).
[10] Kenneth Forbus and Johan de Kleer, *Building Problem Solvers*, MIT Press, 1993.
[11] Mark Hansen, Hakan Yalcin, and John Hayes, 'Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering', *IEEE Design & Test*, **16**(3), 72–80, (1999).
[12] Holger Hoos, 'SAT-encodings, search space structure, and local search performance', in *Proc. IJCAI'99*, pp. 296–303, (1999).
[13] Sajjad Siddiqi and Jinbo Huang, 'Hierarchical diagnosis of multiple faults', in *Proc. IJCAI'07*, pp. 581–586, (2007).
[14] Peter Struss and Oskar Dressler, 'Physical negation: Integrating fault models into the general diagnosis engine', in *Proc. IJCAI'89*, pp. 1318–1323, (1989).

Authors Index