Delft University of Technology
Embedded Software Report Series

Reducing the diagnostic uncertainty of a paper input module by active testing

Alexander Feldman a.b.feldman@tudelft.nl
Arjan van Gemund a.j.c.vangemund@tudelft.nl

Report Number: ES-2009-04
ISSN: 1877-7805

embedded software

TU Delft Delft University of Technology

Published and produced by:
Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Information about the Embedded Software Report Series:
info-es-ewi@tudelft.nl
Information about the Embedded Software Section:
http://www.es.ewi.tudelft.nl/

Reducing the Diagnostic Uncertainty of a Paper Input Module by Active Testing

Alexander Feldman a.b.feldman@tudelft.nl
Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science,
Mekelweg 4, 2628 CD, Delft, The Netherlands

Arjan van Gemund a.j.c.vangemund@tudelft.nl
Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science,
Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract

Model-Based Diagnosis (MBD) algorithms often yield a large number of diagnoses, severely limiting their practical utility. A practical way to deal with such diagnostic uncertainty is to use the active testing approach implemented by Fractal (FRamework for ACtive Testing ALgorithms). Fractal computes a sequence of control settings for reducing the number of diagnoses given a system description and an initial observation. In this report we demonstrate the applicability of Fractal to real-world applications, such as the diagnosis of a Paper Input Module (PIM), part of a heavy-duty printer. In this report we show that (1) Fractal computes intuitive results given the PIM model and an initial observation, (2) Fractal scales to large systems and is computationally efficient, (3) and Fractal leads to a steep reduction in the number of diagnoses even with very coarse-grained models and in the lack of prior failure probabilities. As a result, this report validates the use of Fractal for a range of on-board and off-line diagnostic reasoning tasks.

1. Introduction

The number of diagnoses generated by a Model-Based Diagnosis (MBD) engine (e.g., the General Diagnostic Engine (de Kleer & Williams, 1987)) can be large, and in the worst-case is exponential in the number of components. This ambiguity (uncertainty) of the diagnostic result arises due to to modeling uncertainty (e.g., modeling weakness due to ignorance of abnormal behavior or the need for robustness) and the limited number of observations (sensor-lean systems, limited observation horizons) of typical systems.

Given a set of plausible diagnoses, in certain situations one can devise additional tests (probes), that reduce the diagnostic ambiguity (de Kleer & Williams, 1987). Because most systems, especially those which are not safety-critical (i.e., the production cost is a consideration), are sensor-lean, there is a limited set of test points that can be used to further narrow down the diagnostic solution space. Many systems have built-in testing capabilities with pre-defined test vectors; the drawback with such pre-specified test strategies is that they must cover all possible faults and observations, and so are typically highly sub-optimal.

One way to deal with the large number of diagnoses is to use a novel, *active testing* strategy (Feldman, Provan, & van Gemund, 2009), which *dynamically* determines test vec-

tors from the *entire* test vector space using an MBD approach. In this report we apply Fractal to the real-world scenario of diagnosis a Paper Input Module (PIM). The control assignments computed by Fractal are both intuitive and optimal given the existing model.

Our active testing problem takes as an input a system model, an initial observation, and a diagnosis, and computes the set of input test vectors that minimize diagnostic ambiguity with the least number of test vectors. The test vectors are computed by setting a subset of input settings which, together with the resulting output values, optimally reduce the size of the set of diagnoses.

Computing control settings or probes usually takes place in a larger context. The computational architecture in which we use Fractal is illustrated by Fig. 1. Sequential MBD differs from typical MBD optimization problems in that not all input/output information is available prior the computation. Hence sequential MBD is more akin to non-zero sum games as, usually, there is no utility function to be optimized by the environment. Assuming such function though (worst-case input-output assignments for sequential MBD) is useful as it allows computation of bounds and limit scenarios.

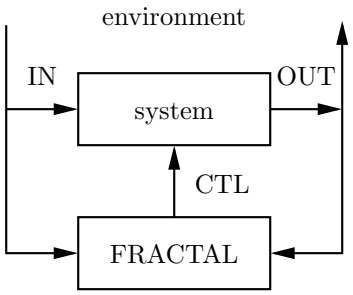

Figure 1: Active testing dataflow for Fractal

Figure 1 shows the interaction of the system with the environment and the computational dataflow of Fractal. The environment provides initial assignments to IN. CTL are all in the default "inactive" state, corresponding to normal system operation. The initial inputs, controls, and outputs (all together) we call initial observation. Fractal then computes a subsequent CTL assignment which combined with new "environmental" inputs IN and outputs OUT (the outputs depend on the injected fault and the inputs) are fed back to Fractal. The process is then repeated until some termination criterion is met. We assume that only Fractal can change the control assignment.

The rest of this report is organized as following. In Sec. 3 we discuss a real-world model to which we apply sequential MBD. In Sec. 4 we introduce the Fractal* algorithm. In Sec. 5 we discuss active testing scenarios with the model from Sec. 3. Further, in Sec. 5, we report on the scalability of Fractal with larger, synthetic models derived from the ISCAS85 benchmark.

2. Related Work

On the application side, there exist related projects in modeling and diagnosing printers. Figure 2, for example, shows the Tightly Integrated Parallel Printer fixture, devised by Palo Alto Research Center (PARC) for studying MBD, probing, testing, and control algorithms.

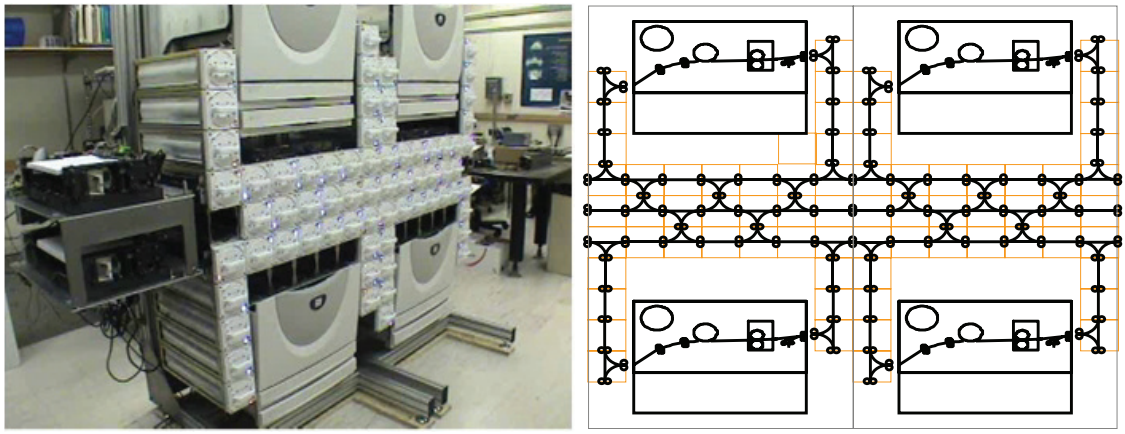

Figure 2: The TIPP fixture (left) and its schematics (right) (photo and diagram courtesy of PARC, used with permission)

A recent approach to active diagnosis is described by Kuhn, Price, de Kleer, Do, and Zhou (2008), where additional test vectors are computed to optimize the diagnosis while the system (a copier) remains operational. Their work differs from ours in that plans (roughly analogous to test sequences) with a probability of failure T are computed statically, and a plan remains unmodified even if it fails to achieve its desired goal (a manifestation of a failure with probability close to T). Conversely, Fractal dynamically computes next-best control settings in a game-like manner.

Other approaches rely on predefined test-vectors, either computed to expose particular faults (ATPG, e.g., (Stephan, Brayton, & Sangiovanni-Vincentelli, 1996)), or selected from a limited test matrix (sequential diagnosis, e.g., (Pattipati & Alexandridis, 1990)).

Early work aimed at diagnostic convergence is the approach by de Kleer and Williams (1987) which computes the probe sequence that reduces diagnostic entropy using a myopic search strategy. Unlike their work, in active testing we assume that probes are not available, other than indirectly exposed through diagnosis based on test vectors, which offers an automated solution.

Generating test vectors to deduce faults has received considerable attention. Automatic test pattern generation (ATPG) aims at verifying particular, single-faults (Stephan et al., 1996). ATPG differs from active testing in that the vectors are specific for particular single-faults, whereas active testing generates a sequence of vectors to isolate *unknown, multiple*-faults, a much harder problem.

Active testing bears some resemblance with sequential diagnosis, which also generates a sequence of test vectors (Pattipati & Alexandridis, 1990; Raghavan, Shakeri, & Pattipati, 1999; Tu & Pattipati, 2003; Kundakcioglu & Ünlüyurt, 2007). The principle difference is that in sequential diagnosis a fault dictionary is used ("fault matrix"). This pre-compiled dictionary has the following drawback: in order to limit the (exponential) size of the dictionary, the number of stored test vectors is extremely small compared to the test vector space. This severely constrains the optimality of the vector sequence that can be generated,

compared to active testing, where test vectors are computed on the fly using a model-based approach. Furthermore, the matrix specifies tests that only have a binary (pass/fail) outcome, whereas active testing exploits all the system's outputs, leading to faster diagnostic convergence. In addition, we allow the inputs to be dynamic, which makes our framework suitable for online fault isolation.

Model-Based Testing (MBT) (Struss, 1994) is a generalization of sequential diagnosis. The purpose of MBT is to compute inputs manifesting a certain (faulty) behavior. The main differences from our active testing approach are that MBT (1) assumes that all inputs are controllable and (2) MBT aims at *confirming* single faulty behavior as opposed to maximally decreasing the diagnostic uncertainty.

Our task is harder than that of Raghavan et al. (1999), since despite the diagnosis lookup using a fault dictionary, the diagnosis task is NP-hard; in our case we compute a new diagnosis after every test. Hence we have an NP-hard sequential problem interleaved with the complexity of diagnostic inference at each step (in our case the complexity of diagnosis is Σ_2^p-hard). Apart from the above-mentioned differences, we note that optimal test sequencing is infeasible for the size of problems in which we are interested.

Brodie, Rish, Ma, and Odintsova (2003) cast their models in terms of Bayesian networks. Our notion of entropy is the size of the diagnosis space, whereas Rish et al. use decision-theoretic notions of entropy to guide test selection.

We solve a different problem than that of Heinz and Sachenbacher (2008) or Alur, Courcoubetis, and Yannakakis (1995). Our framework assumes a static system (plant model) for which we must compute a temporal sequence of tests to best isolate the diagnosis. In contrast, Heinz and Sachenbacher (2008) and Alur et al. (1995) assume a non-deterministic system defined as an automaton.

Esser and Struss (2007) also adopt an automaton framework for test generation, except that, unlike Heinz and Sachenbacher (2008) or Alur et al. (1995), they transform this automation to a relational specification, and apply their framework to software diagnosis. This automaton-based framework accommodates more general situations than does ours, such as the possibility that the system's state after a transition may not be uniquely determined by the state before the transition and the input, and/or the system's state may be associated with several possible observations. In our MBD framework, a test consists of an instantiation of several variables, which corresponds to the notion of test sequence within the automaton framework of Heinz and Sachenbacher (2008).

The sequential diagnosis problem studies optimal trees when there is a cost associated with each test (Tu & Pattipati, 2003). When costs are equal, it can be shown that the optimization problem reduces to a next best control problem (assuming one uses information entropy). In this paper a diagnostician who is given a sequence S and who tries to compute the *next* optimal control assignment would try to minimize the expected number of remaining diagnoses $|\Omega(S)|$.

3. Qualitative PIM Model

We next describe a PIM and introduce a coarse-grained qualitative model of it in Lydia.

3.1 System Overview

The PIM shown in Fig. 3 is used to supply paper to a printing module or to a finisher. Up to three PIM devices can be chained to supply paper to the printing unit and one PIM can be placed between the printer and the finisher to insert blank pages when requested.

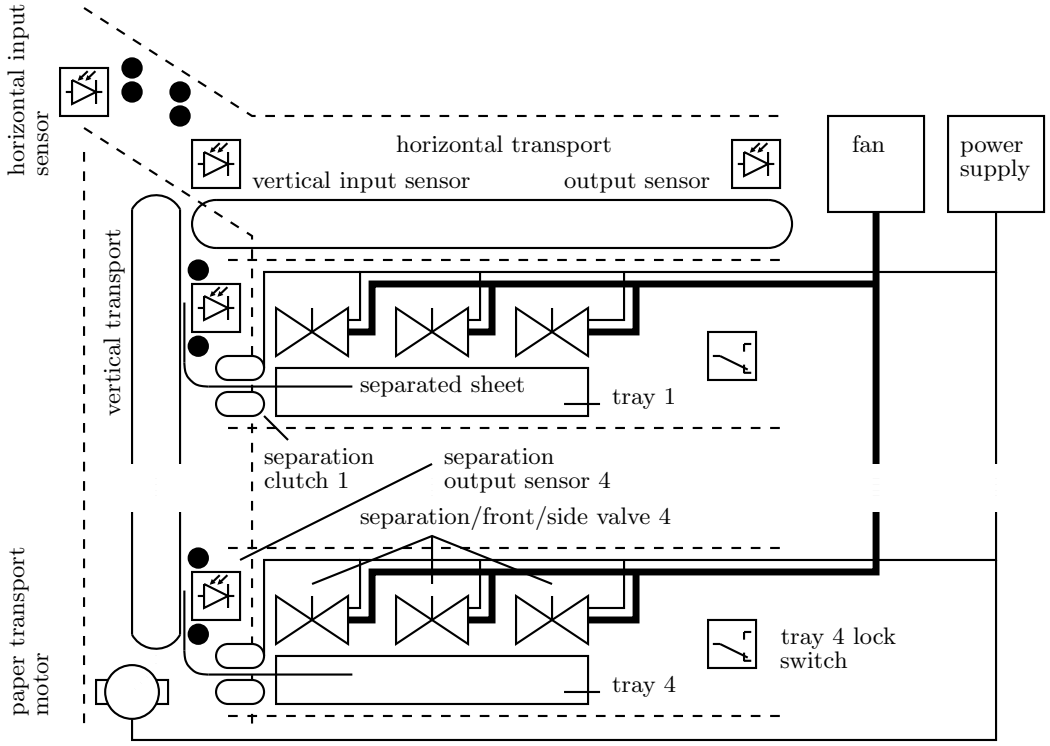

Figure 3: Schematic overview of a PIM

A PIM has four paper trays which are placed vertically and numbered 1 to 4, starting from the top. Each tray can hold some amount of paper. The PIM has its own power supply. It receives commands via a Controller-Area Network (CAN) bus. A command can instruct the PIM to separate a sheet from tray n, horizontally transport a sheet (from a chained tray or from a manual feed), etc.

The PIM uses compressed air to separate the sheets of paper. The air is provided by an electrical fan. A dozen of solenoid valves (three per tray) control the air flow when a separation is requested. After a sheet of paper is separated, it is transported vertically to the top-left part of the PIM (looked from the front) after which the sheet starts its horizontal journey to the output of the PIM.

Each tray (cf. Fig 4) contains a motor which can elevate the paper stack up for separation and down for opening/refilling. There are two sensors which detect the up/down position of the paper stack. In addition to that there is a safety switch for the uppermost position which, under normal conditions, should never detect the tray as the motor should be switched-off after the up sensor reports the completion of an elevation operation.

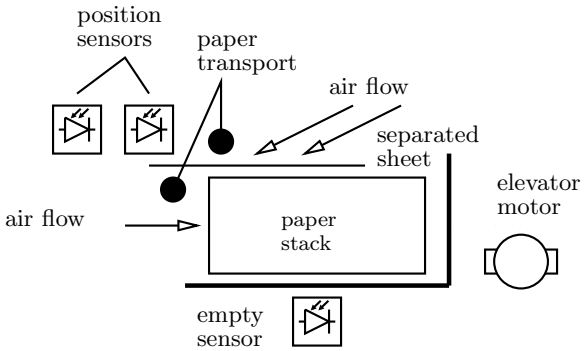

Figure 4: Schematic overview of a paper tray

In addition to the up and down sensors a paper tray has two paper position sensors (in reality the paper tray has also sensors for detecting the paper size but these are not included in the current model). These two position sensors detect a sheet of paper upon leaving the paper tray.

The third and fourth tray, in addition to separation, have a preseparation mode. When commanded to preseparate, the air-valves separate a sheet of paper, its transport starts, the preseparation clutch is connected but the paper is not placed on the belt. This is done to save time for the next separation operation as the sheet from the lower trays has to travel a longer distance in the vertical trajectory.

After a sheet of paper has been separated with compressed air, a clutch connects the transportation of the sheet from the tray to the vertical transport. In the vertical transport subsystem the sheet is placed on the vertical transportation belt (cf. Fig. 3). The vertical transport has sensors for detecting the sheet leaving the tray. The belt in the vertical transport is driven by an electrical motor.

The same motor which drives the vertical transportation belt drives the horizontal transport. The horizontal transport has three sensors. A paper sensor detects a sheet of paper fed to the horizontal input (from a chained PIM or from a manual feed). Another sensor detects a sheet of paper coming from the vertical transport. Finally, the third sensor detects a sheet of paper leaving the device.

In the top-level PIM module are placed the power supply and the fan. The air-duct and the solenoids controlling the air for separation are as well placed in the top-level PIM. Finally, there is a special sensor which detects double sheets.

3.2 PIM Model

We briefly discuss a qualitative PIM model (for a detailed description cf. (Feldman & van Gemund, 2007a)) that supports all queries in the Fractal algorithms. The model is built in the Lydia language (Feldman & van Gemund, 2007b). A Lydia component model of a switch, for example, is shown in Fig. 5. All PIM health variables are of the built-in Boolean type, with a positive literal denoting a healthy state for the respective component and a negative literal denoting the non-healthy state.

```
system Switch(switchValue actual)
{
    bool h;

    switchValue sensed;

    h => (sensed == actual);
}
```

Figure 5: Lydia modeling example

Depending on the desired PIM action (e.g., separate a sheet from tray n or transport horizontally a sheet) each active PIM system is set to its respective state (motors can be on or off, clutches connected or disconnected, etc.). As we will see below, in each state a different set of constraints is applied for inferring the state of the system.

3.2.1 Components

Table 1 provides an overview of all component types which are used in the PIM model.

Table 1: Overview of the PIM component models.

Component	Input	Output	Powered	States	Pr
sensor	-	sensor value	no	-	0.05
switch	-	switch value	no	-	0.05
clutch	paper	paper	yes	connected, disconnected	0.01
motor	-	torque	no	-	0.01
power supply	-	power	no	-	0.01
fan	-	flow	yes	-	0.01
valve	flow	flow	yes	opened, closed	0.025

We model all the components as input/output devices. All components have an output of some (qualitative) type. Some of the components have one or more inputs. The data types for the inputs (if such exist) and outputs of the components are shown in the second and third columns of Table 1. In some of the component models, the electrical power input is considered and this is noted in the fourth column of Table 1. Two of the components (the active components) have more than one state depending on a control value. These sates are listed in the fifth column of Table 1.

The rightmost column of Table 1 shows the fault probability Pr for each component. Note that these probabilities are approximate (de Kleer, 1990) as they are not based on, for example, Failure Modes and Effects Analysis (FMEA) analysis. The Lydia solvers use these probabilities for a more precise partial ordering of the result compared to diagnosis cardi-

nality. These crude probabilities, however, reflect some human diagnostician experience in the likelihood of failure of different components.

We next briefly discuss all component models.

Sensor: We have marked all sensor readings as observable variables. The model stipulates that a sensor is healthy if the observed value is equal to the actual value. Strengthening techniques (e.g., strong-fault models) can reduce the uncertainty inherent to this very coarse-grained model.

Switch: In the PIM abstraction, the Lydia model of a switch is equivalent to the one of a sensor. The naming difference reflects the strict Lydia typing system. All sensors and switches are passive components from the modeling viewpoint. They have only one healthy state, the subsystems which instantiate these sensors and switches provide constraints for the expected values depending on the mode of the PIM.

Clutch: The clutch is an active mechanical device with two states: "connected" and "disconnected". This state is determined by giving a value to a control variable. When a clutch is connected, a paper from the respective tray is fed to the vertical transport belt. A disconnected clutch does not result in a sheet of paper being transported.

In reality, a clutch has no input and output. Indeed, one way to model a clutch is to build a component model with state (e.g., if the clutch is healthy, and it is commanded "connect", and there is an input power it changes its state to "connected"). Such an approach, however, would necessitate the use of constraints in the containing subsystem (in this case the vertical transport) setting the presence of a sheet on the belt depending on the fact if a clutch is connected or not.

A simpler, though arguably physically wrong approach, is to represent a clutch with an input and an output. In this case when the clutch is healthy, commanded "connected", and powered, the output would be the same as the input, i.e., the clutch would transport a sheet of paper. Otherwise, if, for example the mechanism is jammed, there would not be a sheet at the output.

Valve: The PIM uses solenoid valves to direct air when separating sheets. The valve can be commanded to change its state to "opened" or "closed". If the valve is healthy and it is commanded open, then the valve's input flow should equal the valve's output flow. Otherwise, if commanded closed (and the valve not leaking), there should be a lack of output flow. What follows is a model of a valve. Note that this model can be used for a wider variety of valves and devices.

Fan: The fan, needs an electrical power and if it is healthy and commanded on, it produces a positive air flow. A healthy fan, should produce a zero air flow if commanded off.

Motor: A nominally-functioning motor, if commanded on, translates electrical power into torque. The motor should produce a zero torque if commanded off.

Power supply: A special note is needed for the model of the power supply. In the case of the PIM (which lacks redundancy in the power supply) fault reasoning in the case of a fault in the power supply would be impossible to perform by the PIM itself (neither

it would be possible to record observation data). In the PIM model the model of a power supply is extended to, for example, the machinery and cabling used for feeding power to the various components and subsystems. This way, the notion of a power failure can be extended to a failure in the power connector of a valve for example.

The power supply is simply a source of electrical power (we neglect the fact that the power supply is itself connected to the power grid and do not proceed with modeling of the power grid). There is no need to provide a power on-off switch, as we assume that the diagnosis will be done only when the main power supply is switched on and connected to the power main.

3.2.2 Subsystems

The PIM has four subsystems: tray, horizontal and vertical transports and a top-level system. Determining the scope of a subsystem for a model is always subjective and in this model we have tried to follow the mechanical and electrical design decomposition. This is different from the decomposition in the software design, for example, as decomposition decisions there may be driven by bandwidth of communication buses, locality of computational resources, etc.

Each subsystem participates in the paper transport, i.e., it contains mechanical devices, electronics, software, etc., involved in the sheet separation. Hence each subsystem provides at least one input, an output and a health variable. A sheet of paper may jam in the vertical transport for example. The probability of a failure for all the four subsystems is 0.005.

Table 2: Component instantiations

Component	Instantiations	Instantiated By
sensor	28	PIM, horizontal & vertical transports, tray
switch	8	PIM, tray
clutch	6	vertical transport
motor	5	vertical transport, tray
power supply	1	PIM
fan	1	PIM
valve	12	PIM

Table 2 shows the components used by the different subsystems. There are 61 components in total. The seven subsystem instantiations have their own health variables, resulting in a total of 68 health variables. In total the model has 209 variables (of which 43 observables) and we have compiled it to a CNF representation with 605 clauses.

We briefly discuss all subsystems.

Tray: The main function of a paper tray is to hold a stack of sheets. A paper tray also has a motor which elevates the paper stack up for separation, and down for reloading the tray. Up and down sensors and a safety limit switch control this motor. Two separation sensors per tray detect the presence of a sheet when separating. Finally,

an empty sensor detects the presence of a paper stack in the tray. There are many aspects of the tray which we have omitted from the model. In reality, analog sensors determine the paper size, multiple LED indicators show the amount of paper in the tray, etc.

The paper tray is interfaced in the following manner. It receives power from the top-level PIM system. The PIM redistributes the power to its components (in particular to the elevator motor). When commanded to separate a single sheet of paper, the tray provides a sheet of paper to its output (given that the tray holds paper and no fault occurs). The Lydia excerpt below shows the component instantiations and the constraints governing the paper transport of a tray.

Depending on the input control command the tray has to set the states of all active components, it instantiates (active components in the PIM model are motors and clutches). This is done by the constraints that follow (the latter are normally part of the tray system but we show them separately for clarity).

Finally, each mode sets values for the tray sensors and switches. The expected sensor values for each mode are given by the Lydia constraints that follow. These constraints are, as well, part of the tray subsystem (we have discussed the control and sensor constraints outside of the the tray model for clarity).

Vertical transport: The vertical subsystem acts as a demultiplexer for moving a sheet of paper from a tray to the transport belt. After being separated from the tray a sheet is placed on the belt and transported to the output of the vertical subsystem. The rotation of the belt is provided by an electrical motor, the same motor is used to rotate the belt in the horizontal system, hence the model supplies an output of the torque of the motor.

The constraints below control the state of the motor and the separation and pre-separation clutches depending on the value of the "cmdIn" variable.

The vertical subsystem contains multiple sensors for determining the position of the paper. The correct values of all the sensors (in each state) are set by the constraints that follow.

Horizontal transport: The horizontal transport connects the vertical transport (or, alternatively, another pre-stacked PIM) to the output of the PIM. The model of the horizontal transport subsystem is given below. The horizontal transport has three modes: idle, connecting the vertical transport, and connecting the pre-stacked PIM input. The horizontal transport (shown below) is, essentially, a multiplexor. It can transport a sheet either from the vertical transport or from the PIM input. There are sensors at both inputs of the subsystem and at the output.

As is evident from the Lydia excerpt above, depending on the mode of the horizontal transport, it sets the actual sensor values. For example, if the PIM is idle (and the horizontal transport is idle respectively), then the three sensor which are in this subsystem should sense no paper.

The horizontal transport has its own health in addition to the health of the components. In practice, a failure of the sheet to reach the output of the PIM (horizontal

transport) may occur due to a paper jam, due to a mechanical malfunction, etc. In this case the diagnostic system should label the horizontal transport as faulty. Note that this kind of modeling, adds uncertainty to the model, hence it decreases the diagnostic precision.

Top-level system: The top-level PIM system instantiates the horizontal and vertical transports and the four paper trays. It also holds a number of components, in particular the power supply, the fan and a dozen of air valves for separating the sheets. The control part of the PIM top-level system, similar to the horizontal and vertical subsystems, sets the modes of all the top-level subsystems and components.

The top-level system contains a double-sheet sensor and a number of mechanical safety-switches which prevent more than one tray to be ejected at a time. The latter feature ensures the mechanical stability of the PIM. The correct values of the double-sheet sensor and the switches are given by the constraints below.

Note that all the health variables in the PIM model are Boolean. This gives us a hybrid (Boolean health variables, multi-valued constraints) weak-fault model (or model with ignorance of abnormal behavior) for which the Minimal-Diagnosis Hypothesis holds (de Kleer, Mackworth, & Reiter, 1992). As a result the subset minimal diagnoses of the PIM model fully-characterize a diagnostic solution.

4. Algorithm for Active Testing

We continue our discussion with a method for computing control assignments. This method is based on greedy optimization of the expected intersection size, the computation of which is explained in Sec. 4.2. In the following section we use existing Fractal notation and definitions (Feldman et al., 2009).

4.1 Greedy Algorithm for Active Testing

Our AT problem is defined as follows:

Problem 1 (Optimal Control Sequence)**.** Given a system ATS, a sequence (of past observations and controls) $S = \langle \alpha_1 \wedge \gamma_1,\ \alpha_2 \wedge \gamma_2,\ \cdots,\ \alpha_k \wedge \gamma_k \rangle$, and a new observation α_{k+1}, where α_i $(1 \leq i \leq k+1)$ are OBS assignments and γ_j $(1 \leq j \leq k)$ are CTL assignments, compute a new CTL assignment $\gamma^\star$, such that:

$$\gamma^* = \underset{\gamma \in \mathrm{CTL}^\star}{\operatorname{argmin}} E^{\leq}(\Omega^{\cap}(\mathrm{SD}, S), \mathrm{CTL} | \alpha_{k+1} \wedge \gamma) \tag{1}$$

where $E^{\leq}$ denotes the expected remaining number of MC diagnoses which will be explained in Sec. 4.2.

In practice, a diagnostician does not know what the next observation will be. Fully solving an active testing problem would necessitate the conceptual generation of a tree with all possible observations and associated control assignments in order to choose the sequence that, on average, constitutes the shortest (optimal) path over all possible assignments.

The active testing problem can be thought of as an optimization problem. Given an initial control assignment γ we can consider the space of possible control flips. This space

can be visualized as a lattice (Fig. 6 shows a small example). Figure 6 shows the expected number of MC diagnoses for each control assignment. Note that probing can be visualized in a similar way.

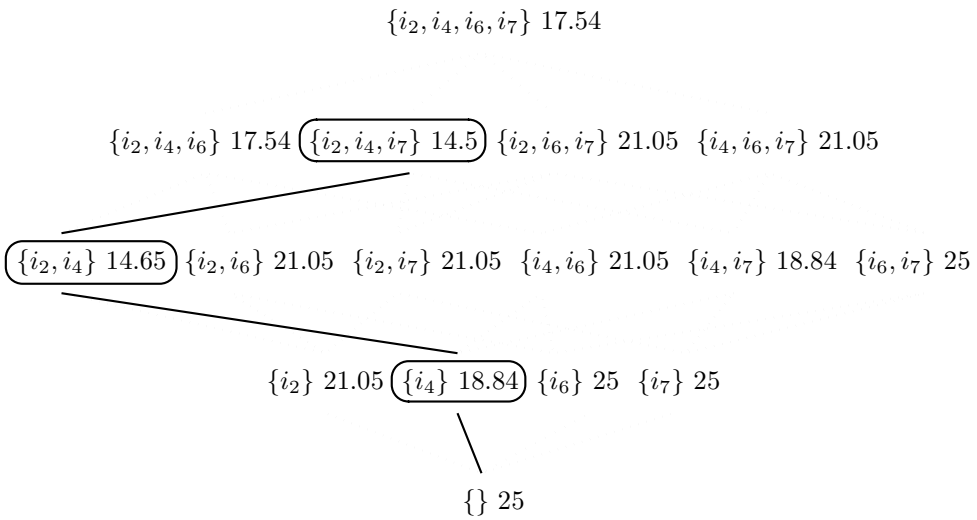

Figure 6: Example of an expectation optimization lattice $(74182, |\text{CTL}| = 4, |\text{IN} = 5|)$

In practice, control literals are mostly independent and even though the space of control assignments is not continuous in general, it has large continuous subspaces. The greedy approach is shown in Alg. 1, which computes a control assignment for a given active testing system and a prior observation.

The set of initial diagnoses is computed from the initial observation in line 2. In line 5, Alg. 1 "flips" the next literal in the current control assignment. The auxiliary FlipLiteral subroutine simply changes the sign of a specified literal in a term. After each "flip" the expected intersection size is computed with a call to Expectation (cf. Alg. 2). If the new expected intersection size is smaller than the current one, then the proposed control assignment is accepted as the current control assignment, and the search continues from there.

While the active-testing problem is worst-case NP-hard (it can be reduced to computing a diagnosis), as we will see in the experimentation section, it is possible to achieve very good average-case performance by choosing an appropriate MBD oracle. The advantage of the greedy approach, in particular, is that the number of computations of the expected number of diagnoses is linear in the number of literals in the control assignment. This is done at the price of some optimality (i.e., the effect of combinations of controls is neglected).

4.2 Expected Intersection Size

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 (pdf) of subsequent outputs (observations), i.e., a subsequent observation α' changes its likelihood. The (non-normalized) a posteriori probability of an

Algorithm 1 Greedy active testing algorithm

```
1: function FRACTAL(ATS, α) returns a control term
       inputs: ATS, active testing system
               α, term, initial observation
       local variables: γ, γ', terms, control configurations
                        E, E', reals, expectations
                        D, set of terms, diagnoses
                        l, literal, control literal
2:     D ← Ω^≤(SD, α)
3:     E ← EXPECTATION(ATS, γ, D)
4:     for all l ∈ γ do
5:         γ' ← FLIPLITERAL(γ, l)
6:         E' ← EXPECTATION(ATS, γ', D)
7:         if E' < E then
8:             γ ← γ'
9:             E ← E'
10:        end if
11:    end for
12:    return γ
13: end function
```

observation α', given an MC operator and an initial observation α is:

$$\Pr(\alpha'|\mathrm{SD},\alpha) = \frac{|\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD},\alpha),\alpha')|}{|\Omega^{\leq}(\mathrm{SD},\alpha)|} \tag{2}$$

The above formula comes by quantifying how a given a priori set of diagnoses restricts the possible outputs (i.e., we take as probability the ratio of the number of remaining diagnoses to the number of initial diagnoses). In practice, there are many α for which $\Pr(\alpha'|\mathrm{SD},\alpha) = 0$, because a certain fault heavily restricts the possible outputs of a system (i.e., the set of the remaining diagnoses in the nominator 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):

$$E^{\leq}(\mathrm{SD},M|\alpha) = \frac{\sum_{\alpha'\in M^*} |\Omega^{\cap}(D,\alpha')| \cdot \Pr(\alpha'|\mathrm{SD},\alpha)}{\sum_{\alpha'\in M^*} \Pr(\alpha'|\mathrm{SD},\alpha)} \tag{3}$$

where $D = \Omega^{\leq}(\mathrm{SD},\alpha)$ and M^* is the set of all possible assignment to the variables in M. Replacing (2) in (3) and simplifying gives us the following definition:

Definition 1 (Expected Minimal-Cardinality Diagnoses Intersection Size)**.** Given a system ATS and an initial observation α, the expected remaining number of MC diagnoses

$E^{\leq}(\mathrm{SD}, \mathrm{OBS}|\alpha)$ is defined as:

$$E^{\leq}(\mathrm{SD}, \mathrm{OBS}|\alpha) = \frac{\sum_{\alpha' \in \mathrm{OBS}^*} |\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha), \alpha')|^2}{\sum_{\alpha' \in \mathrm{OBS}^*} |\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha), \alpha')|} \tag{4}$$

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

The key insight which allows us to build a faster method for computing the expected number of remaining diagnoses is that the prior observation (and respectively the set of MC diagnoses) shifts the probability of the outputs. Hence, an algorithm which samples the possible input assignments (it is safe to assume that inputs are equally likely) and counts the number of *different* observations, given the set of prior diagnoses, would produce a good approximation.

Algorithm 2 Approximate expectation

```
1:  function EXPECTATION(ATS, γ, D) returns a real
        inputs: ATS, active testing system
                γ, term, system configuration
                D, set of diagnoses, prior diagnoses
        local variables: α, β, ω, terms
                         s, q, integers, initially 0
                         Z, set of terms, samples
                         Ê, real, expectation
2:      Z ← ∅
3:      repeat
4:          α ← RANDOMINPUTS(SD, IN)
5:          for all ω ∈ D do
6:              β ← INFEROUTPUTS(SD, OUT, α ∧ γ, ω)
7:              if α ∧ β ∉ Z then
8:                  Z ← Z ∪ {α ∧ β}
9:                  q ← q + |Ω∩(D, α ∧ β ∧ γ)|²
10:                 s ← s + |Ω∩(D, α ∧ β ∧ γ)|
11:                 Ê ← q/s
12:             end if
13:         end for
14:     until TERMINATE(Ê)
15:     return Ê
16: end function
```

Algorithm 2 uses a couple of auxiliary functions: RANDOMINPUTS assigns random values to all inputs and INFEROUTPUTS computes all outputs from the system model, all inputs

```
lydia> diag pim vout1_sensor_failure_separate_tray1
@ ok <diag>
lydia> fm
@ start output <fm>
d1 = { !verticalTransport.seOut[1].h }
d2 = { !separationValve[1].h }
d3 = { !sideBlowValve[1].h }
d4 = { !frontBlowValve[1].h }
d5 = { !fan.h }
d6 = { !h }
d7 = { !verticalTransport.h }
@ stop output <fm>
```

Figure 7: Single sensor failure leading to 7 diagnoses

and a diagnosis.[1] The computation of the intersection size $|\Omega^{\cap}(D, \alpha \wedge \beta \wedge \gamma)|$ can be implemented by counting those $\omega \in D$ for which $\mathrm{SD} \wedge \alpha \wedge \beta \wedge \gamma \wedge \omega \not\models \bot$.

The algorithm terminates when a termination criterion (checked by Terminate) is satisfied. In our implementation Terminate returns success when the last n iterations (where n is a small constant) leave the expected number of diagnoses, $\hat{E}$, unchanged. Our experiments show that for all problems considered, $n < 100$ yields a negligible error.

5. Experimental Results

We next discuss a practical application of Fractal in computing control settings for the PIM. Then we show that Fractal scales to larger systems and results in geometric decay in the number of expected MC diagnoses.

5.1 PIM

Figure 7 shows the results from a scenario in which we have commanded the PIM to separate a sheet from tray 1 and the sheet is not sensed by a vertical transport sensor (Feldman & van Gemund, 2007a). With the existing granularity of the model and the order-of-magnitude prior probabilities for the components, it is not possible to further refine the candidate set than the 7 diagnoses shown in Fig. 7.

Figure 8 shows the output of Fractal, given the initial observation from Fig. 7. In order to validate the experimental results, we need to know the actual fault. This information is missing in the original scenario (our scenarios are artificial as we had no access to a hardware or a software simulator or the physical device). Hence we first randomly choose a fault from the set of faults consistent with the observation (cf. Fig. 7). This actual fault allows us (1) to compute various performance metrics (not discussed in this report) and

1. This is not always possible in the general case. In our framework, we have a number of assumptions, i.e., a weak-fault model, well-formed circuit, etc. The complexity of InferOutputs varies on the framework and the assumptions.

```
fractal> controls pim vout1_sensor_failure_separate_tray1
observation vout1_sensor_failure_separate_tray1: 7 initial diagnoses
injected = { !fan.h }
gamma_1 = { cmdIn = sepTray2 }
fractal> next pim vout1_sensor_failure_separate_tray1
d1 = { !fan.h }
d2 = { !h }
d3 = { !verticalTransport.h }
```

Figure 8: Fractal, initially 7 single-faults

```
lydia> diag pim double_sensor_failure_separate_tray3
@ ok <diag>
lydia> fm
@ start output <fm>
d1 = { !tray[3].sePos2.h, !verticalTransport.seOut[3].h }
d2 = { !tray[3].sePos2.h, !sideBlowValve[3].h }
d3 = { !tray[3].sePos2.h, !frontBlowValve[3].h }
d4 = { !tray[3].sePos2.h, !separationValve[3].h }
d5 = { !tray[3].sePos2.h, !fan.h }
d6 = { !tray[3].sePos2.h, !h }
d7 = { !tray[3].sePos2.h, !verticalTransport.h }
@ stop output <fm>
```

Figure 9: Double sensor failure leading to 7 diagnoses

(2) to verify the correctness of Alg. 1. In this example, a failed-off fan explains the initial observation.

Second, Fractal computes a control assignment which best decreases the number of expected MC diagnoses (in this case single-faults). The resulting control setting is a command to separate sheet from tray 2. Fractal's output is intuitive as in this case, (1) half of the single faults are ruled-out (i.e., they are not consistent with SD and the initial observation) and (2) the remaining 3 single-faults contain the original fault.

Note that in the PIM model there are many dependencies between the control variables, e.g., a command to separate a sheet from tray n implies a command to switch the fan on, a separate command setting the position of each valve, etc. If we require Fractal to compute a position for each of those low-level controls, Fractal would be unable to compute a control assignment decreasing the number of diagnoses due to its greedy nature. In practice however, controls are either independent given the space of possible diagnoses, or they are organized in a hierarchical fashion as in our example.

The scenario shown in Fig. 9 shows an alternative initial-observation leading to 7 double-faults. The result of the Fractal computation is shown in Fig. 10. In this case Alg. 1

```
fractal> controls pim double_sensor_failure_separate_tray3
observation double_sensor_failure_separate_tray3: 7 initial diagnoses
injected = { !tray[3].sePos2.h, !separationValve[3].h }
gamma_1 = { cmdIn = sepTray1 }
fractal> next pim double_sensor_failure_separate_tray3
d1 = { !tray[3].sePos2.h, !verticalTransport.seOut[3].h }
d2 = { !tray[3].sePos2.h, !sideBlowValve[3].h }
d3 = { !tray[3].sePos2.h, !frontBlowValve[3].h }
d4 = { !tray[3].sePos2.h, !separationValve[3].h }
```

Figure 10: Fractal, initially 7 double-faults

computes that the control assignment which optimally reduced the initial candidate space would be a switch from tray 3 to tray 1. Printing from tray 1 results in a nominal observation, which decreases the original set of 7 double-faults to 4 double-faults shown after the "next" Fractal command in Fig. 9. This result cannot be further improved without adding new sensors. The example in Fig. 9 shows that Fractal works with faults of multiple cardinality (many approaches assume single-faults).

The above examples illustrate the intended use of Fractal. Due to the model limitations, however, these examples do not illustrate the full potential of the active testing approach. In what follows we show that (1) Fractal is scalable (applicable to systems with a large number of components) and (2) it results in steep exponential decrease in the number of remaining MC diagnoses.

5.2 ISCAS-85 Experiments

In order to demonstrate the scalability and performance of Fractal on larger systems, we have experimented on the well-known benchmark models of ISCAS85 circuits (Brglez & Fujiwara, 1985). In order to use the same model for *both* MC diagnosis counting and simulation, the fault mode of each logic gate is "stuck-at-opposite", i.e., when faulty, the output of a gate assumes the opposite value from the nominal. In addition to the original ISCAS85 models, we have performed cone reductions as described by Siddiqi et al. (Siddiqi & Huang, 2007).

In addition to the greedy Fractal algorithm (Alg. 1) which we call Fractal*, we have also implemented probing (de Kleer & Williams, 1987). The probing algorithm we call Fractal$^{\mathrm{P}}$. In addition to those two we have also designed and implemented Fractal$^{\mathrm{ATPG}}$, an algorithm that uses ATPG to generate control assignments determining the health of those components that most optimally reduce the diagnostic uncertainty. At the end of this section we will see why Fractal* is our preferred algorithm for active testing.

5.2.1 Experimental Setup

To illustrate the significant diagnostic convergence that is possible, we need initial observations leading to high numbers of initial MC diagnoses. For each circuit, from 1 000 random,

non-masking, double-fault observations, we have taken the 100 with the greatest number of MC diagnoses. The mean number of MC diagnoses varies between 166 (in c432) and 23641 (in c7552).

Given inputs, outputs and controls, Fractal approximates the optimal control settings. The default control policy of Fractal is to apply Alg. 1, a *greedy* policy. We compare it to (1) a *random* policy where at each time step we assign an equally-probable random value to each control and (2) to an *optimal* control policy, where at each time step an exhaustive brute-force search is applied to all control variables.

We define two policies for generating next inputs: *random* and *persistent*. The persistent input policy (when the input values do not change in time) is a typical diagnostic worst-case for system environments which are, for example, paused pending diagnostic investigation, and it provides us with useful bounds for analyzing Fractal's performance. Combining control and input policies gives us six different experimental setups for our framework.

5.2.2 Diagnostic Convergence

The error of Alg. 2 is not sensitive to the number or the composition of the input variables (extensive experimentation with all circuits showed that for $n < 100$ we have $\hat{E}$ within 95% of the true value of E). The value of the expected number of diagnoses $\hat{E}$ approaches the exact value E when increasing the number of samples. Figure 11 shows an example of $\hat{E}$ approaching E for c1908.

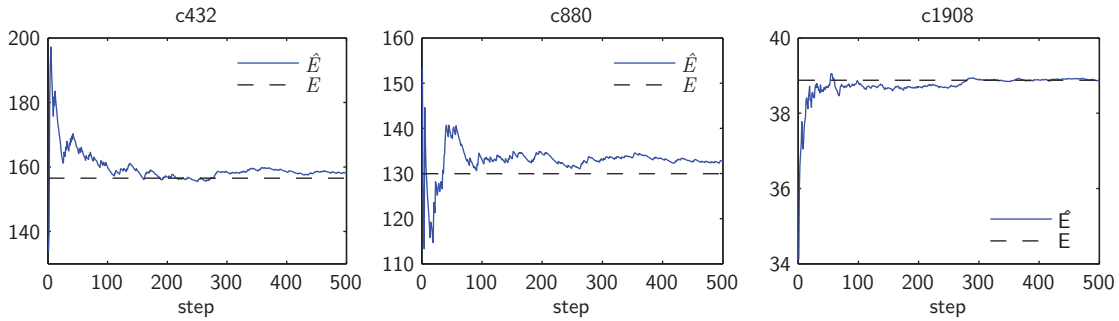

Figure 11: Convergence of $\hat{E}$ with increasing sample size

ISCAS85 has no concept of control variables, hence we "abuse" the benchmark by assigning a fraction of the input variables as controls. It is important to note that turning even a small number of input variables into controls allows for a geometric decay of the diagnostic ambiguity. Figure 12 shows the reduction of the expected number of MC diagnoses as a function of (1) the number of control variables and (2) sequence number. One can observe that a global optimum is reached quickly on both independent axes. Note that Fig. 11 shows an average over 10 pseudo-random experiments for reducing the noise due to the stochastic nature of the algorithm.

Our results show that $\hat{E}$ correlates well with $|\Omega(S)|$. The minimum, maximum, and mean Pearson's linear correlation coefficient for each number of control variables in this example is $\rho_{\min} = 0.897$, $\rho_{\max} = 0.995$, and $\rho_{\text{avg}} = 0.974$.

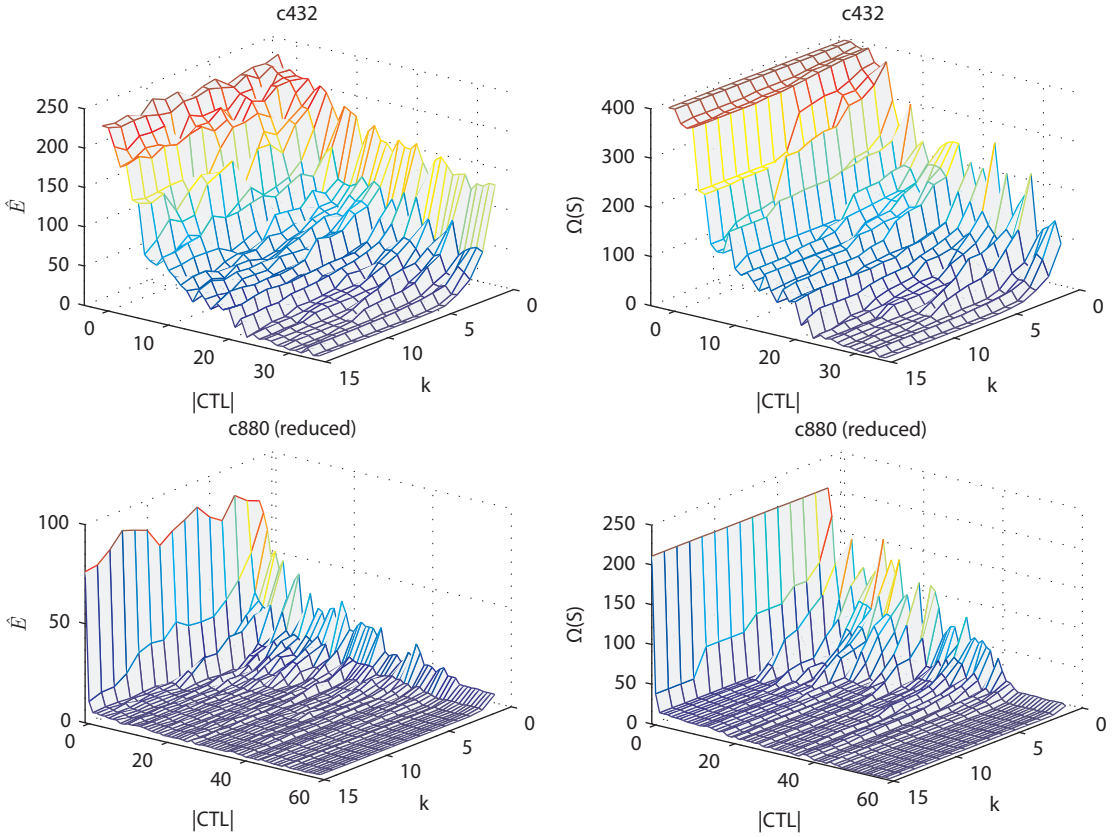

Figure 12: Decay of E (left) and $\Omega(S)$ (right), persistent inputs, FRACTAL*

We have repeated the latter experiment with the reduced c880. The results are similar to the non-reduced c880, and the pairwise correlation coefficients are $\rho_{\min} = 0.68$, $\rho_{\max} = 0.995$, and $\rho_{\text{avg}} = 0.94$. Hence, identification of cones helps the performance of the diagnostic oracle, but does not degrade convergence behavior.

To summarize the effect of the number of controls on the diagnostic convergence, we fit a geometric decay curve $N(k) = N_0 \cdot p^k + N_\infty$ to $\Omega(S)$ for each initial observation and various |CTL|. N_0 is the initial number of diagnoses, N_∞ is the value to which $|\Omega(S)|$ converges, and p is the decay constant. For $p = 0.5$, $N(k)$ halves every step, like in binary search, hence p corresponds to one bit. For $p = 0.25$, p corresponds to two bits, etc.

Table 3 shows the average p for various numbers of control bits $b = \lg|\text{CTL}|$. The goodness-of-fit criterion R^2 for all our experiments is in the range $0.75 - 0.9$, except for a small number of cases in which $R^2 = 1$ due to one-step minimization of $\Omega(S)$. The table demonstrates that the diagnostic convergence generally improves with |CTL|, but that even with small |CTL| better convergence is achieved than in sequential diagnosis ($p = 0.5$ on average).

Similar to our previous experiments we characterize the convergence of the greedy fractal policy by computing the average p over all observations. This time we have computed p for a fixed CTL, where a quarter of the initial inputs are used as controls. The resulting p,

		original			reduced	
id	|IN|	3 bits	4 bits	5 bits	3 bits	4 bits	5 bits
c432	36	0.61	0.69	0.42	0.7	0.71	0.57
c499	41	0.79	0.83	0.77	0.58	0.62	0.52
c880	60	0.5	0.55	0.62	0.49	0.47	0.44
c1355	41	0.71	0.72	0.59	0.8	0.82	0.75
c1908	33	0.68	0.7	0.41	0.54	0.52	0.3
c2670	233	0.45	0.49	0.39	0.39	0.44	0.42
c3540	50	0.39	0.38	0.43	0.79	0.8	0.61
c5315	178	0.52	0.62	0.67	0.81	0.72	0.79
c6288	32	0.31	0.41	0.23	0.64	0.7	0.59
c7552	207	0.62	0.77	0.3	0.59	0.34	0.38

Table 3: Mean p (over all initial observations) for various numbers of control bits and persistent input policies

as well as the goodness-of-fit criterion R^2, for the random and persistent input policies are shown in Table 4.

	original				reduced			
	persistent		random		persistent		random	
id	p	R^2	p	R^2	p	R^2	p	R^2
c432	0.7	0.85	0.62	0.93	0.64	0.88	0.76	0.9
c499	0.81	0.89	0.78	0.89	0.86	0.87	0.78	0.9
c880	0.69	0.9	0.51	0.9	0.83	0.79	0.57	0.94
c1355	0.68	0.81	0.48	0.92	0.81	0.9	0.81	0.89
c1908	0.5	0.89	0.68	0.88	0.69	0.75	0.79	0.81
c2670	0.81	0.91	0.73	0.85	0.64	0.77	0.91	0.88
c3540	0.65	0.8	0.75	0.87	0.72	0.86	0.7	0.9
c5315	0.45	0.82	0.81	0.92	0.84	0.92	0.72	0.84
c6288	0.87	0.83	0.8	0.79	0.7	0.81	0.85	0.91
c7552	0.51	0.88	0.49	0.84	0.79	0.87	0.76	0.82

Table 4: Average (over all observations) p and goodness-of-fit measure for exponential decay best-fit to $\Omega(S)$

5.3 Random and Exhaustive Control Policies and Experimental Summary

The summary of our experiments is best shown in Fig. 13. To factor out sampling error and to be able to perform exhaustive computations we have chosen the smaller 74182 circuit,

which is not part of ISCAS85. The original 74182 (a 4-bit carry-lookahead generator) has 19 components, 9 inputs, and 5 outputs. We have turned 4 of the inputs into controls (hence, $|\mathrm{IN}| = 4$ and $|\mathrm{CTL}| = 4$).

We have considered exhaustive and random control policies in addition to FRACTAL$^{\mathrm{P}}$, FRACTAL$^{\mathrm{ATPG}}$, and FRACTAL*. With exhaustive control the expected number of remaining MC diagnoses is computed at each step and for each possible control combination. This works with 74182 but blows-up with any other larger circuit. With random control policy, at each step, a random value is assigned to each control variable.

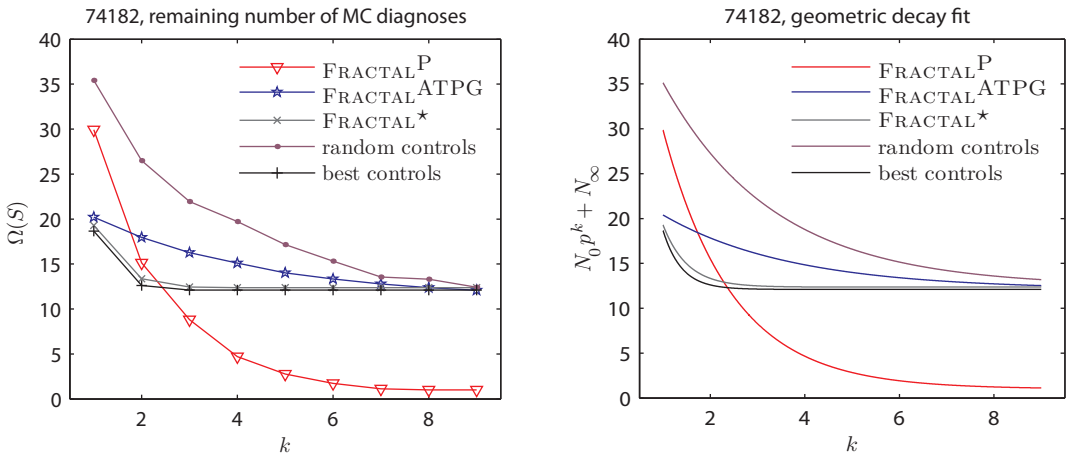

Figure 13: Comparison of all control policies

To reduce the stochastic error when plotting Fig. 13 we have replaced the sampling for computing an expected number of remaining MC diagnoses with an exhaustive method (this is possible as $|\mathrm{IN}| = 5$). The only randomized decision is to choose the actual fault from the initial ambiguity group. To reduce the error due to this stochastic fault injection, we have tested each of the 5 control policies 100 times.

We can see in Fig. 13 that the least informed control policy (the random control policy simply does not use E) shows the worst decay in the number of remaining diagnoses. On the other extreme, the exhaustive control policy achieves the best decay. The price for this policy in terms of computational effort, however, is prohibitive. FRACTAL* achieves decay rates comparable to the exhaustive policy with affordable average-case complexity. FRACTAL$^{\mathrm{ATPG}}$ has better complexity than FRACTAL*, but the whole decay rate curve of FRACTAL$^{\mathrm{ATPG}}$ is bounded from below by the one computed by FRACTAL*.

Probing does not compare to active testing as both approaches have different assumptions on the observability of the model. Figure 13 shows the decay rate of probing to illustrate the different decay curves depending on the observability assumptions. In this experiment the probing decay rate geometric fit with $p = \frac{1}{2}$ almost perfectly fits the actual number of remaining MC diagnoses.

6. Conclusion

In this report we have shown the use of Fractal for solving real-world PIM active testing problems. Fractal has computed control assignments that reduce initial candidate sets containing 7 diagnoses each, to candidate sets containing MC diagnoses of higher cardinality only. Through extensive experimentation with ISCAS85 circuits, we have shown that Fractal works with faults of any cardinality and that Fractal scales to systems containing thousands of components. Fractal efficiently solves complex computational problems by using approximation algorithms which trade-off computational complexity for small decrease in the optimality of the resulting control assignments. As a result of this flexible trade-off, Fractal is suitable for solving both on-line active testing problems as well as larger off-line diagnostic reasoning tasks.

References

Alur, R., Courcoubetis, C., & Yannakakis, M. (1995). Distinguishing tests for nondeterministic and probabilistic machines. In *Proc. ACM Symposium on Theory of Computing*, pp. 363–372.

Brglez, F., & Fujiwara, H. (1985). A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proc. ISCAS'85*, pp. 695–698.

Brodie, M., Rish, I., Ma, S., & Odintsova, N. (2003). Active probing strategies for problem diagnosis in distributed systems. In *Proc. IJCAI'03*, pp. 1337–1338.

de Kleer, J. (1990). Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, *45*(3), 381–291.

de Kleer, J., Mackworth, A., & Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, *56*(2-3), 197–222.

de Kleer, J., & Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

Esser, M., & Struss, P. (2007). Fault-model-based test generation for embedded software. In *Proc. IJCAI'07*.

Feldman, A., Provan, G., & van Gemund, A. (2009). FRACTAL: Efficient fault isolation using active testing. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09), Pasadena, California, USA*.

Feldman, A., & van Gemund, A. (2007a). Building a Lydia model of an océ printer's paper input module. Tech. rep. TUD-SERG-2007-16, Delft University of Technology.

Feldman, A., & van Gemund, A. (2007b). Lydia user guide. Tech. rep. TUD-SERG-2007-016, TU Delft.

Heinz, S., & Sachenbacher, M. (2008). Using model counting to find optimal distinguishing tests. In *Proc. of COUNTING'08*, pp. 91–106.

Kuhn, L., Price, B., de Kleer, J., Do, M., & Zhou, R. (2008). Pervasive diagnosis: Integration of active diagnosis into production plans. In *Proc. DX'08*, pp. 106–119.

Kundakcioglu, O. E., & Ünlüyurt, T. (2007). Bottom-up construction of minimum-cost and/or trees for sequential fault diagnosis. *IEEE Trans. on SMC*, *37*(5), 621–629.

Pattipati, K., & Alexandridis, M. (1990). Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Trans. on SMC*, *20*(4), 872–887.

Raghavan, V., Shakeri, M., & Pattipati, K. (1999). Optimal and near-optimal test sequencing algorithms with realistic test models. *IEEE Trans. on SMC*, *29*(1), 11–26.

Siddiqi, S., & Huang, J. (2007). Hierarchical diagnosis of multiple faults. In *Proc. IJCAI'07*, pp. 581–586.

Stephan, P., Brayton, R., & Sangiovanni-Vincentelli, A. (1996). Combinational test generation using satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, *15*(9), 1167–1176.

Struss, P. (1994). Testing physical systems. In *Proc. AAAI'94*, pp. 251–256.

Tu, F., & Pattipati, K. (2003). Rollout strategies for sequential fault diagnosis. *IEEE Trans. on SMC*, *33*(1), 86–99.