Third International Diagnostic Competition – DXC'11

Scott Poll[1], Johan de Kleer[2], Rui Abreu[3], Matthew Daigle[6], Alexander Feldman[4], David Garcia[2], Alberto Gonzalez-Sanchez[5], Tolga Kurtoglu[2], Sriram Narasimhan[6], and Adam Sweet[1]

[1] *NASA Ames Research Center, Moffett Field, CA, 94035, USA*
scott.poll@nasa.gov adam.sweet@nasa.gov

[2] *Palo Alto Research Center, Palo Alto, CA, 94304, USA*
dekleer@parc.com dgarcia@parc.com kurtoglu@parc.com

[3] *Faculty of Engineering, University of Porto, Porto, Portugal*
rui@computer.org

[4] *Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud, CH-1401 Yverdon-les-Bains, Switzerland*
alexander.feldman@heig-vd.ch

[5] *Delft Univiersity of Technology, Delft, The Netherlands*
a.gonzalezsanchez@tudelft.nl

[6] *University of California, Santa Cruz @ NASA Ames Research Center, Moffett Field, CA, 94035, USA*
sriram.narasimhan-1@nasa.gov matthew.j.daigle@nasa.gov

ABSTRACT

We present the third implementation of a framework created jointly by NASA Ames Research Center, Palo Alto Research Center, and Delft University of Technology to compare and evaluate diagnosis algorithms (DAs). This year's competition, DXC'11, introduces a software track in addition to the industrial and synthetic tracks of previous competitions. A total of eleven DAs competed in the three tracks. The paper describes the systems, diagnostic problems of the tracks, fault scenarios, evaluation metrics, participating DAs, results and analysis.

1 INTRODUCTION

The problem of detecting and isolating faults in physical systems has led to various solution approaches including expert, model-based, data-driven, and stochastic reasoning methods. However, there have been few efforts to evaluate and compare these different approaches in a standardized way. NASA Ames Research Center (ARC), Palo Alto Research Center (PARC), and Delft University of Technology decided to combine efforts to create a generalized framework that would establish a common platform to evaluate and compare diagnosis algorithms (Kurtoglu *et al.*, 2009a). The objectives for developing this framework are to accelerate research in theories, principles, and computational techniques for monitoring and diagnosis of complex systems; to encourage the development of software platforms that promise more rapid, accessible, and effective maturation of diagnostic technologies; and to provide a forum that can be utilized by algorithm developers to test and validate their technologies on real-world physical systems.

The First International Diagnostic Competition (DXC'09) was the first implementation of the above-mentioned framework (Kurtoglu *et al.*, 2009b). The overall goal of the competition was to systematically evaluate diagnostic technologies and to produce comparable performance assessments for different diagnostic methods. DXC'09 pitted 12 diagnosis algorithms competing in 3 different tiers on 2 different tracks (industrial and synthetic). In each tier, the DAs were provided a description of the system being diagnosed and sample data sets consisting of nominal and faulty scenarios. Several metrics that covered timing, technical, and computational performance were computed and a single final ranking score was calculated to determine the winners in each tier.

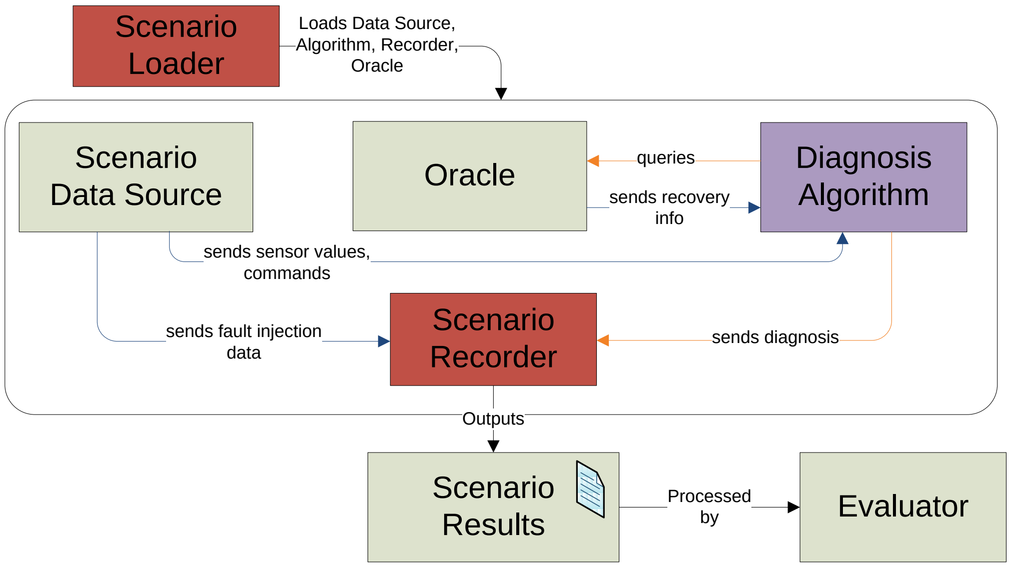

Figure 1: DXF run-time architecture.

Based on feedback from DXC'09, several changes were made for the Second International Diagnostic Competition, DXC'10 (Poll *et al.*, 2010). The primary change was in the evaluation criteria. In DXC'09 we used several metrics that were consolidated into a single ranking score. In order to accurately represent real-world problems, DXC'10 defined use cases indicating what the role of the diagnosis results would be. For the industrial track a decision support use case was chosen, where the diagnosis result would be used to decide what recovery action(s) should be performed in order to minimize mission costs, which may include loss of mission objectives and/or loss of vehicle. We also added incipient faults and intermittent faults to one of the diagnostic problems to make it more challenging. For the synthetic track a troubleshooting use case was chosen, where internal variables were not directly observable and probes were needed to gather more information. The goal was to correctly identify the faults with the fewest probes.

The Third International Diagnostic Competition (DXC'11) includes the same diagnostic problems and use cases for the industrial and synthetic tracks as DXC'10. This year's competition also features, for the first time, a software track. The goal of this track is to provide common ground to evaluate techniques that diagnose failures in software systems. We used the framework as a basis to diagnose software systems used by the software engineering community. In particular, the programs used in this track were taken from both the Siemens and Software Infrastructure Repository (SIR) benchmark programs. This year, and as a starting point (we would very much like to open the track to other algorithms in the future), the track was dedicated to techniques based on coverage information. Hence, the DAs have to pinpoint the failure root cause using a program statement's coverage information only.

The paper is organized as follows. Section 2 gives a quick review of the DXC framework. Section 3 describes the diagnostic problems that were presented to the competitors. Section 4 lists the kinds of faults that were injected. Section 5 explains how the evaluation was performed. Section 6 presents the results. Section 7 concludes the paper.

2 DXC FRAMEWORK

The DXC framework allows systematic evaluation and comparison of diagnosis algorithms under identical experimental conditions. The key components of this framework include representation languages for the physical system description, sensor data and diagnosis results, a runtime architecture for executing DAs and diagnostic scenarios, and an evaluation component that computes performance metrics based on the results from diagnosis algorithm execution.

We provide a summary of the DXC framework (DXF) in this paper and refer the reader to (Kurtoglu *et al.*, 2009a; Feldman *et al.*, 2010) for additional details. Figure 1 shows an overview of the DXC software components and the primary information flows. All communication is ASCII-based, and all the modules communicate via TCP ports by using a simple message-based protocol. Next, we provide a brief description of each software component.

Scenario Loader: Executes the Scenario Data Source, Recorder, and Diagnosis Algorithm. Scenario Loader ensures system stability and clean-up upon scenario completion. This is the main entry point for performing a diagnostic experiment.

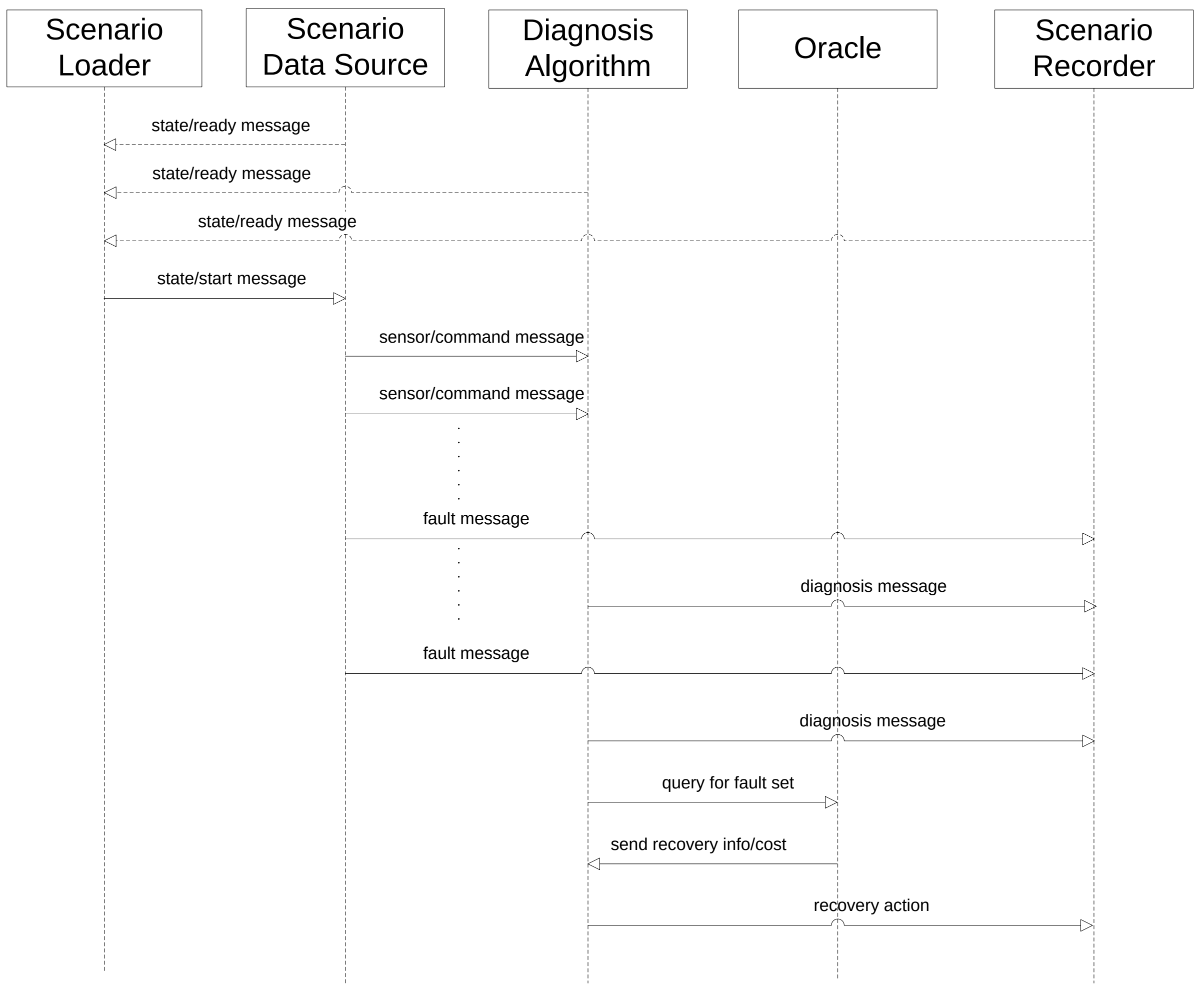

Figure 2: Diagnostic session sequence diagram.

Scenario Data Source: Provides scenario data from previously recorded datasets. The provenance of the data (whether hardware or simulation) depends on the system in question. A scenario dataset contains sensor readings, commands (note that the majority of classical MBD literature does not distinguish commands from observations), and fault injection information (to be sent exclusively to the Scenario Recorder). Scenario Data Source publishes data following a wall-clock schedule specified by timestamps in the scenario files.

Scenario Recorder: Receives fault injection data and diagnosis data into a scenario results file. The results file contains a number of time-series which are used by the evaluation module for the computation of metrics. Scenario Recorder is the main timing authority, i.e., it timestamps each message upon arrival before recording it to the results file.

Diagnosis Algorithm: A DA receives sensor and command data, performs diagnosis, and sends the diagnosis results back. As long as the DAs comply to the provided API, there are no restrictions on a DA; for example, a DA may read precompiled data, or use external (user supplied) libraries, etc.

Diagnostic Oracle: The Diagnostic Oracle is only relevant to the Industrial Track. It provides a querying capability to the DAs in one of two ways: 1) it takes a diagnostic output produced by a DA and returns the lowest cost action(s) associated with the provided diagnosis, or 2) it takes a diagnostic output and specific actions produced by a DA and returns the corresponding cost.

Evaluator: Takes scenario result file and applies metrics to evaluate DA performance. The metrics and evaluation procedures are detailed in Section 5.

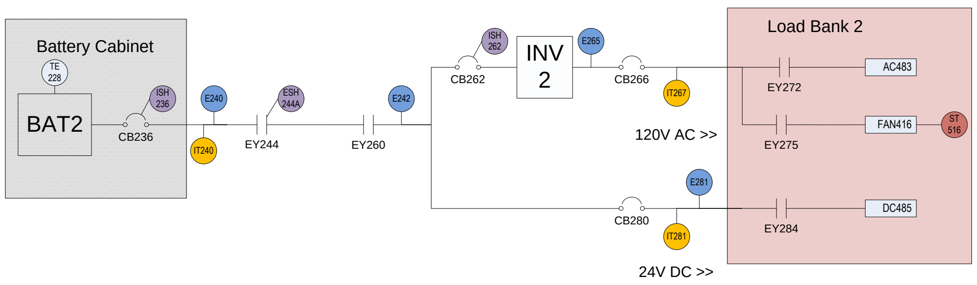

Figure 3: ADAPT-Lite system for diagnostic problem I.

Figure 2 shows a diagnostic session sequence diagram that shows how the different components interact.

3 DIAGNOSTIC PROBLEMS

Four diagnostic problems were announced for DXC'11, two industrial track problems (DP-I, DP-II), one synthetic (DP-III), and one software (DP-IV). There were no entries for the second diagnostic problem of the industrial track so we omit discussion of DP-II. Refer to (Poll *et al.*, 2010) for a description of diagnostic problem II and results from DXC'10.

3.1 Industrial Track

The hardware system used for the industrial track is the Electrical Power System testbed in the ADAPT lab at NASA Ames Research Center (Poll *et al.*, 2007). A subset of the testbed defines the system ADAPT-Lite, which is shown in Figure 3.

Table 1: DP-I characteristics

Aspect	**DP-I**
System	ADAPT-Lite
Operational scenario	Single-string UAS mission
Diagnostic use case	Abort recommendation
Number of components	25
Number of modes	102
Initial relay state	Closed
Initial circuit breaker state	Closed
Nominal mode changes	No
Multiple faults	No
Fault types	Discrete
	Offset
	Drift (incipient)
	Intermittent offset

Diagnostic problem I is the same as DP-I in DXC'10. The problem mimics the use of the ADAPT-Lite system in a single-string Unmanned Aircraft System (UAS) mission. The primary objective of the diagnosis algorithm in this operational scenario is to provide decision support to a remote pilot or an autonomous controller by making an "abort" (land the vehicle) recommendation.

The correct recommendation for a scenario depends on the injected failure mode and, for certain failure modes, the fault parameters. Any failure which cuts off power to any of the three loads results in an abort recommendation. Other failure modes may lead to degraded performance which can be tolerated if the fault magnitude is below some threshold. For these scenarios, giving the correct recommendation requires isolating the failure mode and estimating the fault parameter(s).

Table 1 summarizes the main characteristics of diagnostic problem I. With the sensors provided, there are four ambiguity groups: (i) AC483 failed off and EY272 stuck open, (ii) FAN416 failed off and EY275 stuck open, (iii) DC485 failed off and EY284 stuck open, and (iv) INV2 failed off and CB262 failed open. In each case however, the recovery action is the same for both faults in the ambiguity group. Figure 4 illustrates the parametric fault profiles used in ADAPT-Lite.

3.2 Synthetic Track

Diagnostic problem III builds on the synthetic track for DXC'09. Table 2 summarizes the ISCAS85 digital circuits used for this problem. The task corresponds to the use case where internal variables are not directly observable and additional, expensive information needs to be gathered to isolate the fault.

The DA is provided a system, input vector, and output vector manifesting the injected fault(s). The task of the DA is to gather more information (via internal probes) to correctly identify the injected faults. We presume all probes are of equal cost. DAs were limited to 20 seconds CPU time (although in most scenarios the DAs took less than 1 CPU second).

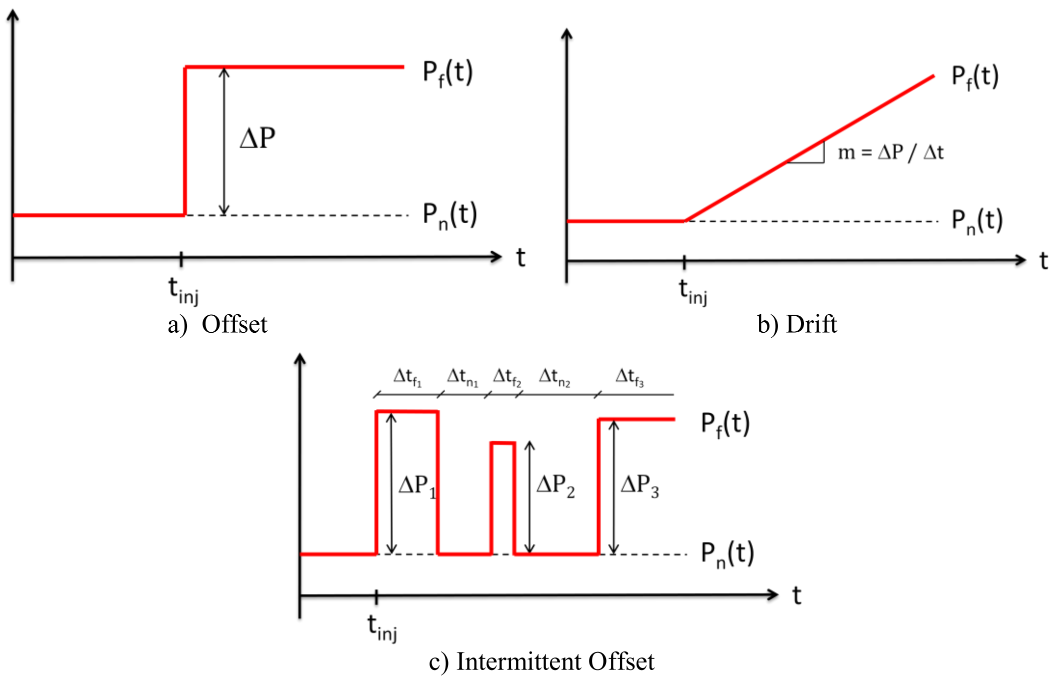

Figure 4: Diagnostic problem I fault profiles.

Note that for many tasks of MBD (e.g., computing Max-Fault Min-Cardinality (MFMC) observations (Feldman *et al.*, 2008)), the number of components in the ISCAS85 circuits can be reduced by performing cone identification (Siddiqi and Huang, 2007; de Kleer, 2008). The number of components in the reduced circuits is shown in the rightmost column of Table 2. We left the decision to identify cones to the competitors, i.e., we distribute the non-reduced circuits. We inject all faults at one instant. For example, if 3 components are faulted, the first observation provided to the DA is the result of all three faults injected simultaneously.

Table 2: ISCAS85 circuits for DP-III

	original				reduced
sys	|IN|	|OUT|	|COMPS|	variables	|COMPS|
74182	9	5	19	47	6
74L85	11	3	33	77	15
74283	9	5	36	81	14
74181	14	8	65	144	15
c432	36	7	160	356	59
c499	41	32	202	445	58
c880	60	26	383	826	77
c1355	41	32	546	1133	58
c1908	33	25	880	1793	160
c2670	233	140	1193	2695	167
c3540	50	22	1669	3388	353
c5315	178	123	2307	4792	385
c2688	32	32	2416	4684	1456
c7552	207	108	3512	7232	545

3.3 Software Track

The diagnostic problems of the software track are taken from both the Siemens and Software Infrastructure Repository (SIR) benchmark programs. In particular, for DXC'11, we use the well-known software programs in the Siemens benchmark set as well as flex, grep, gzip, sed, and space as taken from SIR. They are all coded in the C programming language, and can be obtained from

Table 3: Subject programs for DP-IV

Program	**LOC**	**Tests**	**Program Type**
print_tokens	539	4,130	Lexical Analyzer
print_tokens2	489	4,115	Lexical Analyzer
replace	507	5,542	Pattern Recognition
schedule	397	2,650	Priority Scheduler
schedule2	299	2,710	Priority Scheduler
tcas	174	1,608	Altitude Separation
tot_info	398	1,052	Information Measure
space	9,126	500	ADL Parser
gzip	6,708	211	Compressor
sed	9,014	184	Stream Editor
grep	13,287	809	String Matching
flex	14,194	107	Lexer Generator

the SIR's website (http://sir.unl.edu). Every program provides a set of test inputs, which were created with the intention of providing full branch test coverage. Table 3 provides some additional information about the subject programs (LOC is lines of code).

4 FAULT INJECTION AND SCENARIOS

Test scenarios were created for use in the DXC framework. Scenarios included faults of various types and complexity. We describe the general characteristics of the scenarios and faults in the industrial, synthetic and software tracks in this section.

4.1 Industrial Track Scenarios

Experimental scenarios of approximately four minutes in length were acquired using the ADAPT Electrical Power System (EPS) testbed for diagnostic problem I. The testbed allows for the repeatable injection of faults into the system in three ways: hardware-induced faults (e.g., turning off inverters, tripping circuit breakers, manipulating loads); software-induced faults (e.g., sending extraneous relay commands or blocking intended relay commands); introduction of faulty components (e.g., inserting a burned out light bulb). The first two methods were used for DXC'11.

Diagnostic problem I includes 154 scenarios previously used in DXC'10 and 73 new scenarios. There are 30 nominal and 197 single-fault scenarios. The parametric fault profiles illustrated in Figure 4 are injected for sensor faults as well as AC and DC load faults. For the latter, a programmable electronic load is used to vary the AC and DC load resistances. Of the 227 scenarios, 96 are "abort" and 131 are "no-abort" cases. Table 4 summarizes the faults and scenarios used for DP-I.

Table 4: DP-I faults

Type	Subtype	Fault	#	abort / no-abort
nominal		no fault	30	0 / 30
circ. breaker		failed open	6	6 / 0
inverter		failed off	2	2 / 0
load	fan	failed off	2	2 / 0
		over speed	2	2 / 0
		under speed	1	0 / 1
	AC load	resistance offset	11	5 / 6
		resistance drift	12	6 / 6
		intermittent res. offset	11	5 / 6
		failed off	1	1 / 0
	DC load	resistance offset	12	6 / 6
		resistance drift	13	6 / 7
		intermittent res. offset	12	6 / 6
		failed off	2	2 / 0
relay		stuck open	7	7 / 0
sensor	position	stuck	3	0 / 3
	current, temp., voltage	offset	31	13 / 18
		stuck	11	6 / 5
		drift	26	10 / 16
		intermittent offset	32	11 /21
		Totals:	227	96 / 131

4.2 Synthetic Track Scenarios

The synthetic track benchmark includes 1074 scenarios. Each scenario consists of an input vector, an injected fault (hidden from the DA), an output vector and all internal values (hidden from the DA but which the framework uses to supply values for any probe requests from the DA). The benchmark has many very high-cardinality (> 20) injected faults.

Constructing a good set of benchmark scenarios has many challenges. If we presume components fail independently, a fair sampling would primarily yield low cardinality faults. Or, if we included faults which yield no observable symptom for the input vector, there would be no possibility for the DA to isolate the correct fault. Or, if we included faults which had no causal influence on the output observation, most DAs would be penalized for not finding such faults. Therefore, we have two principles for constructing these scenarios: (1) every injected fault produces a symptom (often called 'strong' faults) and (2) every injected fault causally affects the output vector.

The scenarios are generated as follows. We generate scenarios with ever increasing fault cardinality. For each scenario, we generate a random input vector and then compute the correct corresponding output vector. We randomly choose some output vector bits to flip. For each such pair we compute one minimum cardinality (minc) diagnosis. If the minc is not the desired cardinality we flip a random input/output vector bit, and follow the gradient to find an input/output vector pair which yields the desired minc. In most cases, there is a very large ambiguity group. We generate 10,000 candidates of that ambiguity group and randomly choose one as the injected fault. Generating a benchmark in this way takes days of CPU time.

Table 5: Example injected DP-IV faults

Type	Nominal	Faulty
Relational	`a < b`	`a <= b`
	`a > b`	`a >= b`
	`a == b`	`a != b`
Arithmetic	`a + b`	`a - b`
	`a * b`	`a / b`
Increment	`a++`	`++a`
	`a--`	`--a`
Indexing	`a[i]`	`a[i+1]`
	`a[i]`	`a[i-1]`
Assignment	`a = b;`	`a;`

4.3 Software Track Scenarios

Although versions with seeded faults are provided with the programs in the Siemens/SIR benchmark sets, we do not use them in the competition. Instead, we seed random faults in the program by randomly altering the source code (a technique known as *mutation*). For each program we generate 25 scenarios with a single fault, and 25 scenarios with multiple faults. The faults we introduce affect only the 'functional' statements. We change additions into subtractions, relational operators (less than into less or equal than), etc. We do not alter 'static' code, e.g., function prototypes, data types, variable declarations, etc. Several example injected faults are presented in Table 5.

The software diagnostic competition is based on a *regression testing* scenario. We assume a developer has committed a new revision of the subject program, which has to be tested for defects (regressions). The DA must choose test inputs from the test pool to execute and obtain a diagnosis. The DA is given 30 seconds to select tests, send probe commands, and return a diagnosis to the scenario loader.

5 EVALUATION

Using the scenarios described in the previous section, the DAs were executed using the DXC framework on similar hardware within each track and evaluated using a set of metrics.

5.1 Industrial Track Metrics

Diagnosis algorithms were ranked using a decision cost metric, M_{dc}. The decision cost is the cost incurred had the DA's recommendation been acted upon. Additionally, we compute the metrics used in DXC'09.

Table 6: DP-I decision costs, M_{dc}

DA rec. \ Actual Case	abort	non-abort
abort	0	$c_{mission}$
non-abort	$c_{mission} + c_{vehicle}$	0

Table 7: DXC'09 metrics summary

Metric	Name	Class
M_{fd}	fault detection time	detection
M_{fn}	false negative scenario	detection
M_{fp}	false positive scenario	detection
M_{da}	scenario detection accuracy	detection
M_{fi}	fault isolation time	isolation
M_{err}	classification errors	isolation
M_{cpu}	CPU load	computation
M_{mem}	memory load	computation

For DP-I, the two main categories of costs are cost of losing the vehicle and cost of not completing the mission. In this use case the DA is only responsible for deciding if a mission should be aborted or not. Hence there are 4 outcomes (2 answers from the DA versus 2 actual situations). Let the cost of losing the mission be $c_{mission}$ and the cost of losing the vehicle be $c_{vehicle}$. Table 6 shows the costs incurred in each of the 4 possible outcomes. For DP-I $c_{mission}$ is set to 25, $c_{vehicle}$ is set to100.

The metrics used for evaluating DAs in DXC'09 are summarized in Table 7. Please see (Kurtoglu *et al.*, 2009; Feldman *et al.*, 2010) for detailed definitions and related discussion. Note that DXC'09 metric M_{ia} has been renamed M_{err} in the table. The metrics in the table are per scenario metrics. To calculate "per system" metrics an unweighted average is taken over all scenarios and indicated with an overbar.

5.2 Synthetic Track Metrics

For DP-III, we used the following metric to score the result for each DA:

$$n + E[t] + E[\bar{t}]$$

where n is the number of probes the DA made before it reported fault isolation, and $E[t] + E[\bar{t}]$ is the expected number of tests needed to identify the correct diagnosis when the DA isolates the wrong diagnosis as defined in (Feldman *et al.*, 2010). In many cases the DA isolates the correct diagnosis and $E[t] + E[\bar{t}] = 0$. This metric assumes the cost of a measurement is the same as the cost of testing an individual component, which is not valid in general.

5.3 Software Track Metrics

For DP-IV, DAs were ranked according to two main metrics: probing cost (M_{probe}), and residual diagnostic effort (M_{cd}). M_{probe} measures how many tests had to be executed to obtain the definitive diagnosis, with a cost c_{test}=1. M_{cd} measures the quality of the final diagnosis assuming a developer would inspect the suspect candidates top to bottom in the ranking. M_{mem} and M_{cpu} were also recorded for DP-IV algorithms.

5.4 Computing platform

DP-I diagnosis algorithms were evaluated using the DXC framework on two computers with identical hardware (Intel® Pentium™ 2x3GHz, 2 GB RAM), one running Windows™ and the other Linux. AntigenDX was run on Linux, QED and QED-PC were run on Windows. DP-III algorithms were evaluated on one processor core of an Intel XEON™ 6x3.33GHz, 12GB RAM, running Linux. DP-IV algorithms were evaluated on a dual processor Intel XEON 8x2.93GHz, 32 GB RAM, running Linux.

6 RESULTS

Using the evaluation approach described in the previous section, we computed metrics and rankings for the eleven diagnosis algorithms that participated in the Third International Diagnostic Competition.

6.1 Diagnosis Algorithms

The teams that participated in the industrial, synthetic and software tracks are listed in Table 8. In what follows we provide a brief description of each DA.

1. AntigenDX: A real-time, two-tier system which first utilizes Artificial Immune System (AIS) models to detect and categorize sensor faults, then aggregates all detected sensor-level faults and associated characteristics and, through a rule-based expert system, identifies the electrical system component most likely to have produced those faults, as well as estimates of the significant parameters of its failure (Mange *et al.*, 2011).
2. QED: A model-based diagnosis system based on qualitative event-based fault isolation. Statistically significant deviations of measured from model-predicted values imply the presence of faults. These deviations are abstracted into symbolic event-based descriptions of fault-induced behavior, which are compared to predicted event sequences to isolate faults. Fault identification uses quantitative methods to compute fault parameters and further refine fault hypotheses (Daigle and Roychoudhury, 2010).

Table 8: DXC'11 participating DAs

DA	DP	Algorithm Type
AntigenDX	I	AIS + Rule-based
QED	I	Model-based, global
QED-PC	I	Model-based, local
NGDE-NP	III	Model-based
NGDE-E	III	Model-based
NGDE-F	III	Model-based
Optimist	III	Edge case DA
Probee	III	Edge case DA
Ochiai	IV	Similarity coefficient
Raptor-H	IV	Model-based + Bayes
Raptor-EPS2	IV	Model-based + Bayes

3. QED-PC: Similar to QED, but uses the Possible Conflicts diagnosis approach (Pulido and Alonso-Gonzalez, 2004). The global system model is decomposed into minimal submodels containing a sufficient analytical redundancy to generate fault hypothesis from observed measurement deviations. (Daigle *et al.*, 2011).
4. NGDE-NP: An Allegro Common Lisp implementation of the classic GDE. NGDE (de Kleer, 2011) uses an improved minimum-cardinality candidate generator to construct diagnoses from conflicts. It performs no probes, and returns a probability weighted set of diagnoses (max 1000).
5. NGDE-E: An elaboration of NGDE-NP. After determining the minc (< 1000) diagnoses, uses a minimum entropy technique to determine the best probe to make next. Once a single minc candidate is isolated it verifies this candidate is the actual fault with more probes.
6. NGDE-F: Identifies one minc candidate and attempts to verify it.
7. Optimist: Always returns all components working correctly.
8. Probee: Probes all measurable points to determine a candidate.
9. Ochiai: A statistics-based algorithm for software fault diagnosis. It takes as input component involvement in the execution of a given test case and whether or not the test case has passed. The diagnostic report yielded by Ochiai is a ranked list of components likely to be responsible for observed failures (Abreu *et al.,* 2011).
10. Raptor-H: A DA that combines test selection and diagnosis, both using a spectrum-based model. Test selection is done through an ambiguity reduction approach, similar to (Gonzalez-Sanchez *et al.*, 2011). The outcome of the selected test and covered source code statements are then passed to the diagnosis algorithm. The diagnosis algorithm is based on a minimal hitting set computation, followed by a Bayesian update that takes advantage

Table 9: Diagnostic problem I metrics

DA	M_{dc}	$\overline{M}_{fd}$ (s)	$\overline{M}_{fn}$	$\overline{M}_{fp}$	$\overline{M}_{da}$	$\overline{M}_{fi}$ (s)	$\overline{M}_{err}$	$\overline{M}_{cpu}$ (ms)	$\overline{M}_{mem}$ (kb)
AntigenDX	350	62.8	0.005	0.009	0.987	62.8	34	680	1935
QED	50	10.5	0.010	0.004	0.987	124.1	22	824	5356
QED-PC	375	15.2	0.020	0.026	0.956	123.9	38	589	5320

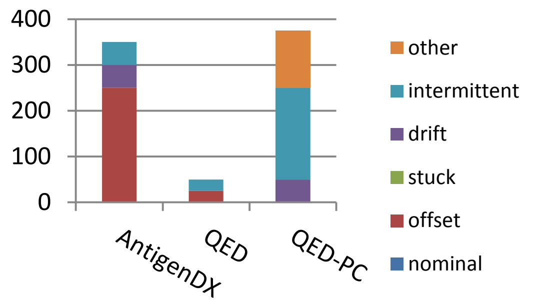

Figure 5: DP-I cost breakdown by scenario fault type.

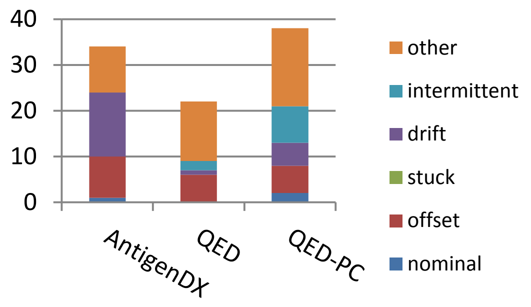

Figure 6: DP-I classification error breakdown by scenario fault type.

of precompiled fault intermittency information (Gonzalez-Sanchez *et al.*, 2011).

11. Raptor-EPS2 is similar to Raptor-H, but the Bayesian diagnosis step uses fault intermittency information derived from the current set of observations, using the EPS2 policy (Abreu *et al.*, 2009).

6.2 Industrial Track Results

The metrics for DP-I are shown in Table 9; diagnosis algorithm QED had the lowest cost and is the winner of DP-I. For comparison, a DA that always recommends abort would have received a cost of 131 * 25 = 3275; a DA that always recommends no-abort would have received a cost of 96 * 125 = 12,000. The DAs had significantly lower cost scores than participants in DXC'10. This is somewhat expected given that more than 2/3 of the test scenarios are from DXC'10 and were available for validating the diagnostic approach.

AntigenDX incurred costs of 25 on 4 scenarios for recommending an abort when not required and 125 on 2 scenarios for not recommending when required; QED incurred costs of 25 for 2 no-abort scenarios, it always recommended abort when needed. QED-PC incurred costs of 25 for 5 no-abort scenarios and 125 for 2 abort scenarios.

We show the breakdown of costs by fault type for each DA in Figure 5. Offset, drift, and intermittent faults include hardware (AC483, DC485) and sensor (e.g., IT267, IT281, etc.) fault injection scenarios. Category "other" includes circuit breaker, inverter, fan, and AC and DC load failed-off fault scenarios. None of the DAs incurred any costs for nominal scenarios.

AntigenDX and QED had the same detection accuracy, with AntigenDX having a lower false negative rate while QED had a lower false positive rate. As mentioned for DXC'10, the fault detection and isolation times are noticeably higher than DXC'09 because of the need to accumulate more evidence in the case of drift and intermittent fault types.

Figure 6 shows the breakdown of classification errors by fault type. In a scenario, the number of classification errors is the number of misclassified components. Ruling out guessing, a perfect DA would have 12 classification errors, all in the category "other", because of the ambiguity groups identified in section 3.1.

6.3 Synthetic Track Results

The results for DP-III are shown in Table 10. The Probee DA uses minimal CPU time and its score is completely determined by the number of internal nodes of the system. It always identifies the correct diagnosis. The Optimist DA uses minimal CPU time and its score is completely determined by the non-probe portion of the metric. NGDE-NP does no probing, so its score is also purely determined by the non-probe factor of the metric. NGDE-E and NGDE-F both selectively probe so their total score is determined by the sum of the probes and the utility penalty of the metric. All CPU figures are in seconds. All numbers in the table are averages for each system.

Probee is the worst strategy, followed by Optimist. NGDE-NP performs a bit better than these edge-case DAs, but not significantly. NGDE-E and NGDE-F perform similarly on simpler examples. The difference

Table 10: Diagnostic problem III scores

System	Probee (CPU<0.01)	Optimist (CPU<0.01)	NGDE-NP		NGDE-E				NGDE-F			
	Probes, Score (E=0)	E, Score (Probes=0)	E, Score (Probes=0)	CPU	Probes	E	Score	CPU	Probes	E	Score	CPU
74182	14	14	5.0	0.03	2.5	0	2.5	0.28	2.5	0	2.5	0.02
74L85	30	21	13.8	0.07	3.5	0	3.5	0.22	3.5	0	3.5	0.02
74283	31	25	21.8	0.10	7.4	0	7.4	0.50	7.8	0	7.8	0.07
74181	57	47	35	0.04	8.1	0	8.1	0.55	8.1	0	8.1	0.07
C432	153	118	96	0.08	12.4	0	12.4	0.79	15.8	0	15.8	0.21
C499	170	169	146	1.06	13.9	17.4	31.3	3.1	16.9	0	16.9	0.3
C880	357	306	260	2.2	16.2	38.2	54.4	4.5	17.5	0	17.5	0.30
C1355	514	439	365	4.0	11.8	88.5	100.3	5.6	13.4	0	13.4	0.36
C1908	855	639	571	0.69	11.6	13.1	24.7	6.6	10.2	0	10.2	0.89
C2670	1129	1043	705	3.5	12.5	258	270.5	6.3	14.7	12.9	27.6	1.3
C3540	1647	1094	602	0.47	7.6	0	7.6	2.24	3.2	0	3.2	0.47
C5315	2184	1969	1464	2.9	14	319.8	333.8	6.9	12.5	0	12.5	0.71
C6288	2384	1208	1149	7.4	5.25	443.9	449.15	10.6	6.4	331	337.4	6.4
C7552	3405	2244	1725	1.1	4.2	0	4.2	2.0	2.6	0	2.6	0.98

becomes apparent for more complex scenarios. NGDE-E would perform best on all scenarios given infinite CPU time as it uses the theoretically optimum strategy. However, generating enough candidate diagnoses needed to determine the optimum probe can exceed the CPU limit. In those cases, it returns its current best estimate of the diagnoses. It pays a very significant metric penalty for its incorrect diagnoses. NGDE-F performs on average the best as making probes are relatively inexpensive and greatly improves the CPU time to compute diagnoses. NGDE-F significantly reduces the utility component of the metric – far more than the slight increase in probes. NGDE-F is a far simpler method as its strategy of confirming component failure by simply measuring its io-behavior avoids more complex minimum entropy evaluation of possible probe points. The table suggests a better strategy would be a mix: use NGDE-E for smaller examples and NGDE-F for more complex ones.

6.4 Software Track Results

We show the averaged metrics for each of the three DAs in Table 11. Both Raptor-H and Raptor-EPS2 have much lower probing costs than Ochiai. Please note that this is an unfair comparison since Raptor includes a test selection heuristic, whereas Ochiai just requests every possible test.

Besides a lower probe cost, Raptor-H and Raptor-EPS2 have lower residual diagnostic effort as well. Per

Table 11: Diagnostic problem IV metrics

DA	$\overline{M}_{probe}$	$\overline{M}_{cd}$	$\overline{M}_{mem}$ (kb)
Ochiai	109200	1138	3114
Raptor-H	26413	862	8804
Raptor-EPS2	11056	889	10320

program, Raptor-H performs better in the cases where the precomputed intermittency information has a low error. On average, however, there is little difference between both algorithms.

Finally, when it comes to memory consumption, Ochiai is the winner since it is a much lighter-weight approach than Raptor-* Bayesian diagnosis. The DX framework was not able to measure CPU time reliably and therefore the results are not given in this paper.

Figure 7 provides a detailed breakdown of the M_{cd} per program. For the large programs (flex, grep, gzip, sed, space) the results of Raptor-H are better than Raptor-EPS2. On the other hand, for the other, smaller programs, Raptor-EPS2 surpasses Raptor-H. Raptor-EPS2 requires a larger number of observations than Raptor-H to obtain a good diagnosis. Since the number of observations is limited by the number of tests that Raptor is able to select in 30 seconds, on big programs Raptor-H has an advantage.

7 CONCLUSION

We presented the implementation of the Third International Diagnostic Competition, DXC'11. We were pleased to introduce the software track in this year's competition and have DAs in all three tracks. A number of people expressed interest in participating in DXC'11 and submitted placeholder algorithms but unfortunately many had to withdraw from the competition for varying reasons. The participants in the industrial track were competitors in DXC'10 as well and showed greatly improved performance this year.

We hope that this work can be continued moving forward by identifying new physical systems and new diagnostic problems. The DXC framework provides an easy way to create an evaluation platform for new problems. Our sincere hope is that the framework is adopted by a growing number of people and applied to a wide

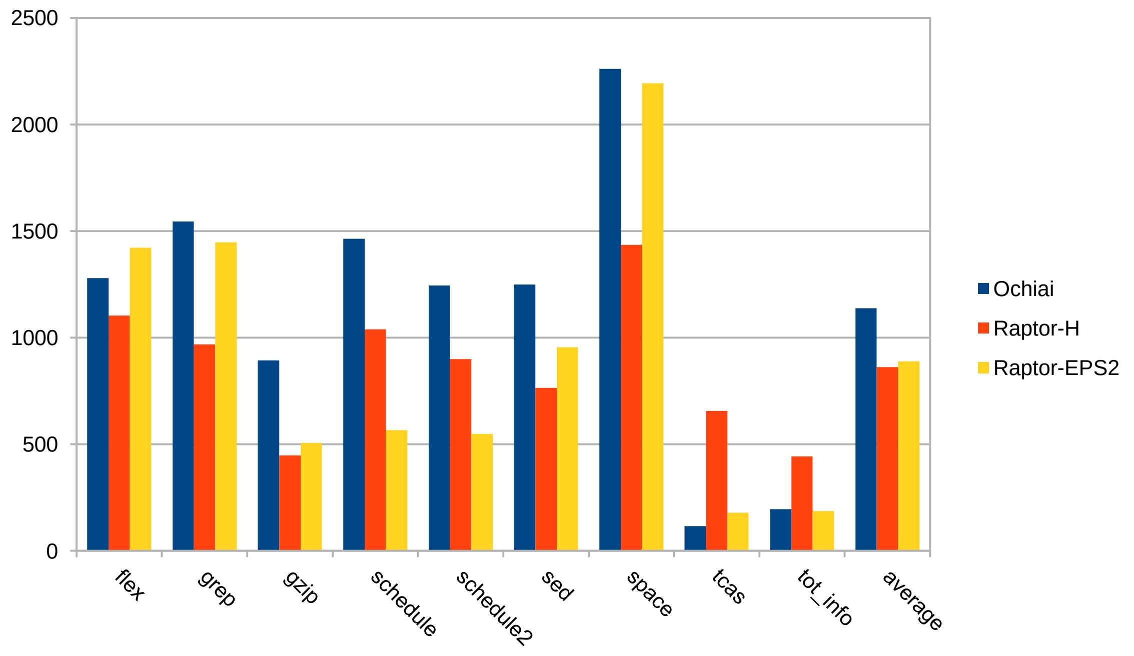

Figure 7: DP-IV residual diagnosis effort breakdown by program.

variety of physical systems including diagnosis algorithms from several different research communities. The long-term goal is to create a database of performance evaluation results which will allow system designers to choose the appropriate DA for their system given the constraints and metrics in their application.

ACKNOWLEDGMENT

We extend our gratitude to Shekhar Gupta (PARC), Callie Kwiatkowski (University of Wisconsin – Milwaukee), David Nishikawa (NASA), all DXC'11 competitors, and many others for valuable discussions, criticism and help.

REFERENCES

(Abreu *et al.* 2009) R. Abreu, P. Zoeteweij, A.J.C. van Gemund. A New Bayesian Approach to Multiple Intermittent Fault Diagnosis. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI'09)*, pp. 653--658, Pasadena, CA, USA, July 2009.

(Abreu *et al.,* 2011) R. Abreu, P. Zoeteweij, A.J.C. van Gemund. Simultaneous Debugging of Software Faults. In *Journal of Systems and Software (JSS)*, vol. 84(4), pp. 573-586, Elsevier, 2011.

(Daigle and Roychoudhury, 2010) M. Daigle and I. Roychoudhury. Qualitative Event-based Diagnosis: Case Study on the Second International Diagnostic Competition. In *Proceedings of 21st International Workshop on Principles of Diagnosis*, Portland, OR, 2010.

(Daigle *et al.*, 2011) M. Daigle, A. Bregon, and I. Roychoudhury. Qualitative Event-based with Possible Conflicts: Case Study on the Third International Diagnostic Competition. In *Proceedings of 22nd International Workshop on Principles of Diagnosis*, Munich, Germany, 2011.

(de Kleer and Williams, 1987) J. de Kleer and B. C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97-130, 1987.

(de Kleer, 2008) J. de Kleer. An Improved Approach for Generating Max-Fault Min-Cardinality Diagnoses. In *Proceedings of 19th International Workshop on Principles of Diagnosis*, Blue Mountains, Australia, 2008.

(de Kleer, 2011) J. de Kleer. Hitting Set Algorithms for Model-based Diagnosis. In *Proceedings of 22nd International Workshop on Principles of Diagnosis*, Munich, Germany, 2011.

(Feldman *et al.*, 2008) A. Feldman, G. Provan, A. van Gemund. Computing observation vectors for Max-Fault Min-Cardinality diagnoses. In *Proc. AAAI'08*, pp. 919–924.

(Feldman *et al.*, 2010) A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, A. van Gemund. Empirical Evaluation of Diagnostic Algorithm Performance Using a Generic Framework. In *International Journal of Prognostics and Health Management, Vol. 1 (2)*, 2010.

(Gonzalez-Sanchez *et al.*, 2011) A. Gonzalez-Sanchez, R, Abreu, H.G. Gross, A.J.C. van Gemund. Prioritizing Tests for Fault Localization through Ambiguity Group Reduction. In *Proceedings of the 26th International Conference on Automated Software Engineering (ASE'11)*. Lawrence, KA, November 2011. To appear.

(Kurtoglu *et al.*, 2009a) 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 *Proceedings of 20th International Workshop on Principles of Diagnosis*, Stockholm, Sweden, 2009.

(Kurtoglu *et al.*, 2009b) T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, A. Feldman. First International Diagnosis Competition – DXC'09. In *Proceedings of 20th International Workshop on Principles of Diagnosis*, Stockholm, Sweden, 2009.

(Mange *et al.*, 2011) J. Mange, D Daniszewski, and A. Dunn. Artificial Immune Systems for Diagnostic Classification Problems. In *Proceedings of 21st International Workshop of Principles of Diagnosis*, Munich, Germany, 2011.

(Mosterman and Biswas, 1999) P. J. Mosterman and G. Biswas. Diagnosis of Continuous Valued Systems in Transient Operating Regions. In *IEEE Trans. on Systems, Man and Cybernetics, vol. 29, no. 6*, pp. 554-565, Nov. 1999.

(Narasimhan and Brownston, 2007) S. Narasimhan and Lee Brownston. HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis. In *Proceedings of 18th International Workshop on Principles of Diagnosis*, Nashville, TN, 2007.

(Poll *et al.*, 2010) S. Poll, J. de Kleer, A. Feldman, D. Garcia, T. Kurtoglu, and S. Narasimhan. Second International Diagnostic Competition – DXC'10. In *Proceedings of 21st International Workshop on Principles of Diagnosis*, Portland, OR, 2010.

(Pulido and Alonso-Gonzalez, 2004) B. Pulido and C. Alonso-Gonzalez. Possible conflicts: a compilation technique for consistency-based diagnosis. *IEEE Trans. Syst. Man Cy. B.*, 34(5):2192–2206, 2004.

(Siddiqi and Huang, 2007). S. Siddiqi and J. Huang. Hierarchical Diagnosis of Multiple Faults. In *Proc. IJCAI'07*, pp. 581–586.