Gradient Boosting-accelerated Evolution for Multiple-Fault Diagnosis

Hongfei Wang[1,2,5], Chenliang Luo[1,2,5], Deqing Zou[1,2,5], Hai Jin[1,3,5], Wenjie Cai[4,5]
[1]National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab,
[2]Hubei Engineering Research Center on Big Data Security,School of Cyber Science and Engineering,Hubei Key Laboratory of Distributed System Security,
[3]Cluster and Grid Computing Lab, School of Computer Science and Technology, [4]College of Public Administration,
[5]Huazhong University of Science and Technology, Wuhan 430074, China e-mail:{hongfei, chenliangluo, deqingzou, hjin, wenjiecai}@hust.edu.cn

***Abstract*—Logic diagnosis is a key step in yield learning. Multiple faults diagnosis is challenging because of several reasons, including error masking, fault reinforcement, and huge search space for possible fault combinations. This work proposes a two-phase method for multiple-fault diagnosis. The first phase efficiently reduces the potential number of fault candidates through machine learning. The second phase obtains the final diagnosis results, by formulating the task as an combinational optimization problem that is later iteratively solved using binary evolution computation. Experiments shows that our method outperforms two existing methods for multiple-fault diagnosis, and achieves better diagnosability (improved by 1.87×) and resolution (improved by 1.42×) compared with a state-of-the-art commercial diagnosis tool.**

***Index Terms*—logic diagnosis, machine learning, gradient boosting, evolution computation**

I. Introduction

Defects are almost inevitable during chip manufacturing [1]. Identifying root causes of the failing chips is critical to help correct design bugs and rectify manufacturing process. Logic diagnosis finds the potential defective sites, usually with failure behavior described by fault models. An ideal diagnosis means accurate fault callouts with good resolution, expected to effectively guide physical failure analysis (PFA) for detailed micro-inspection of the failures. However, PFA and similar hardware-based measures may still fail to identify the failure culprits due to technical reasons (such as spatial limitation from the infrared light), in addition to their slow and expensive nature. Therefore, software-based logic diagnosis is indispensable. To expedite time-to-market and makes profits, quality diagnosis is highly desirable to ramp up the yield during the early bring-up stage.

Fabs today keeps integrating an increasingly huge number of transistors into a tiny silicon chip, expecting better performance, power, and area (PPA). One side-effect is that multiple faults are more likely to occur within a single chip. Multiple-fault diagnosis is difficult due to the interaction among faults, such as *error masking* and *fault reinforcement*. Error masking means a fault, which can be propagated to one or more primary outputs (POs) alone, is canceled out in the present of other faults [2]. Fault reinforcement means a fault, whose propagation path is originally blocked, can be observed on the POs in the present of other faults [3]. Enumerating all possible fault combinations would guarantee an optimal solution, but is inherently impractical due to the exponential search space, thus has rarely been used.

Multiple fault diagnosis attracts much attention. One approach focuses on improvement of controllability and observability by design for diagnosis [4]. The downside of this method is that it may sacrifice chip's PPA. Another way is to generate additional test patterns for diagnosis at the cost of extra ATPG and testing efforts [5]. X-fault model is proposed to prevent error masking, which is able to examine all the possible combinations of faults [6]. However, it does not consider error reinforcement and makes no use of the information from passing patterns. Whereas passing information is indeed exploited in [7] to analyze error reinforcement based on the propagation paths, misprediction may happen because each time only a single minimal set of faults is targeted. The assumption that multiple faults can be described by a minimal set of fault candidates does not guarantee to hold. The work in [8] considers both error masking and fault reinforcement simultaneously by scoring each candidate based on its fault effects. Then it iteratively picks the top-ranking candidates for different multiple fault combinations, causing false candidates to escape. The work in [9] studies transition delay faults that tend to cluster in a small region. It scores each candidates with single-fault simulation and chooses any candidate that has a higher score than the threshold. A heuristic method is proposed to examine all the possible fault sites using particle swarm optimization (PSO) [10], but it may suffer from scaling issues when the size of the circuit grows.

Machine learning (ML) provides an alternative approach to deal with multiple fault diagnosis. One previous work [11] analyzes failing pattern types and then uses random forests [12] to identify true candidates. However, the forest predictor is tuned based on the F1-score that weighs recall and precision the same, which may mistakenly eliminate true candidate given the number of false candidates is much larger than that of true ones. We will show in our experiment that our method is better than the random forests based approach. Another work [13] uses two deep learning models, developed from single and two-faults injected training dataset separately, to estimate the number of faults and then determine the true candidates. It is limited to two faults injected for a multiple fault, however.

In this work, we propose a method called **GEM** (**G**radient boosting-accelerated **E**volution for **M**ultiple-fault diagnosis). Unlike [11] which determines the subset of true candidates that

This research is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62172173 and No. 62372198. Deqing Zou is the corresponding author.

979-8-3503-4859-0/DATE24/© 2024 EDAA

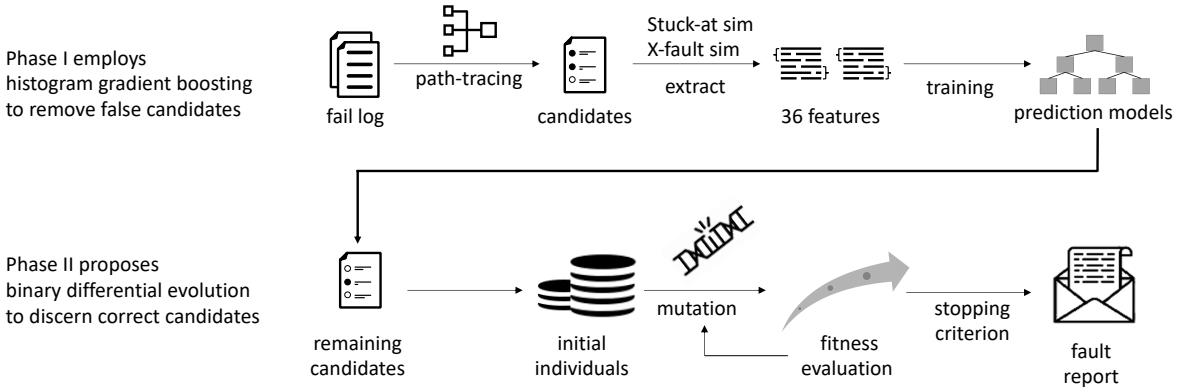

Fig. 1. Work flow of GEM: Gradient boosting-accelerated Evolution for Multiple-fault diagnosis.

may cause overkill, GEM uses machine learning to remove the false candidates, retaining as many true candidates as possible. Both stuck-at fault simulation and X-fault simulation are employed to obtain failing data, expecting that more failing behavior can be observed. Then we utilize a differential evolution based metaheuristic for the set-cover problem. It uses failing patterns and passing patterns to pinpoint combinations of candidates that can jointly explain failing responses and to remove false candidates respectively.

II. Two-Phase Multiple-Fault Diagnosis Method

A. Methodology Overview

Mathematically, multiple-fault diagnosis can be formulated as a binary optimization problem [14], defined as

$$\max_{X \in \mathbb{B}^m} f(X). \tag{1}$$

X is a binary vector such that $X = \{X_j\}_{1 \le j \le m} \in \mathbb{B}^m$. m is the number of candidates, i.e., all possible faults after fault collapsing. The binary variable X_j indicates the presence or absence of the j^{th} candidate. The objective function $f(\cdot)$ measures how simulation response matches tester response. The larger $f(X)$ is, the more likely X describes the true multiple faults.

Many combinatorial optimization methods can be used to solve Eq. (1), as it is a set covering problem in nature. Evolution algorithms are among the solution menu, such as the work uses particle swarm optimization (PSO) [10] for multiple-fault diagnosis. They can be slow and inaccurate due to the huge search space that needs to be throughly explored before arriving at an optimal one, however.

GEM is a two-phase methodology for multiple-fault diagnosis, illustrated in Fig. 1. Phase I produces the maximum possible number of true candidates in an efficient manner. Phase II is a fine-tuning one, which is expected to find optimal solution in an shrunk solution space, determined by Phase I. Specifically, Phase I of GEM begins by back-tracing from each failing POs to obtain the initial candidates. Next, machine learning is used to remove most of the false ones, reducing the input scale for the follow-on analytics. Phase I learns a conservative model that retains as many true candidates as possible, tolerating some false candidates classified as true ones. Phase II of GEM then formulates the problems as an binary optimization one that can be iteratively solved by a variant of differential evolution strategy, with adaptation in this work.

B. Phase I: Histogram Gradient Boosting

Phase I begins by applying a path-tracing procedure described in [15] to the failing circuits. For each failing POs of a failing pattern, path-tracing is performed to locate the candidates, i.e., potential faulty sites with and their corresponding stuck-at polarities. Next, GEM uses ML to filter out false candidates. To enable this data-driven process, a total of 36 features corresponding to each candidate are extracted. Table I lists 18 features, each one of which is calculated for both stuck-at simulation and X-fault simulation [6], thus obtaining $18 \times 2 = 36$ ones.

In Table I, for a candidate under consideration: an "explainable output" means simulation response equals to tester response on that PO; an "explainable pattern" means all the POs are "explainable"; an "reachable output" means the site corresponding to the candidate is structurally reachable from the PO; an "related pattern" means the candidate can be found by performing path tracing on the pattern.

With the obtained data features, GEM leverages gradient boosting [16] [17] to build classification models that differentiate between false candidates and true ones. Gradient boosting refers to a family of ML methods that learn ensembles of prediction models, which usually are decision and classification trees that are incrementally trained and added to the ensemble for improving prediction accuracy. The specific one used is histogram gradient boosting (HGB) [17] , most notably for its efficiency.

Suppose there is a dataset $D = \{(\mathbf{x}_j, y_j)\}$, where $|D|$ is the number of candidates, $\mathbf{x}_j$ is a 36×1 feature vector extracted for the j^{th} candidate. The binary label $y_j = 1$ if and only if the j^{th} candidate truly exists. K base predictors $P_k(\cdot)$ are trained for the HGB ensemble. HGB predicts $\mathbf{x}_j$ as

$$\hat{y}_i = \textstyle\sum_{k=1}^{K} P_k(\mathbf{x}_j) \tag{2}$$

TABLE I
DESCRIPTION OF EXTRACTED FEATURES

features	description
1	ratio of explainable patterns to all the failing patterns
2	ratio of explainable outputs to all the outputs
3	the max ratio of explainable outputs to all the outputs in one failing pattern
4	ratio of explainable patterns to all the failing patterns, considering only reachable outputs
5	ratio of explainable outputs to all the reachable outputs
6	the max ratio of explainable outputs to all the reachable outputs in one failing pattern
7	ratio of explainable patterns to all the failing patterns, considering only reachable failing outputs
8	ratio of explainable outputs to all the outputs, considering only reachable failing outputs
9	the max ratio of explainable outputs to all the reachable failing outputs in one failing pattern
10	ratio of explainable patterns to related failing patterns
11	ratio of explainable outputs to all the outputs in related failing patterns
12	the max ratio of explainable outputs to all the outputs in one related failing pattern
13	ratio of explainable patterns to all related failing patterns, considering only reachable outputs
14	ratio of explainable outputs to all the reachable outputs in related failing patterns
15	the max ratio of explainable outputs to all reachable outputs in one related failing pattern
16	ratio of explainable patterns to related failing patterns, considering only reachable failing outputs
17	ratio of explainable outputs to all reachable failing outputs in related failing patterns
18	the max ratio of explainable outputs to all the reachable failing outputs in one related failing pattern

To learn a base predictor $P_k(\cdot)$, HGB minimizes the following objective loss function L_k with regularization,

$$L_k = \sum_{j=1}^{|D|} L(y_i, \hat{y}_i) + \gamma T + \frac{\lambda}{2} \sum_{t=1}^{T} w_{kt}^2 \quad (3)$$

During training, the loss function $L(\cdot,\cdot)$ measures the erroneous inconsistency between the prediction $\hat{y}_i$ and the true label y_i, in this binary classification problem. γ and λ are two regularization factors. T is the number of leaf nodes and w_{kt} is the value of leaf node t. By employing a second-order Taylor expansion, L_k is approximately simplified as

$$L_k = \sum_{t=1}^{T} (G_{kt} w_{kt} + \frac{1}{2}(H_{kt} + \lambda) w_{kt}^2) + \gamma T \quad (4)$$

where G_{kt} and H_{kt} is the the first and the second derivative of the loss function L_k, respectively. HGB greedily selects splitting points by maximizing the decrease of objective function,

$$\triangle L_j = \frac{1}{2} \frac{G_L^2}{H_L + \lambda} + \frac{1}{2} \frac{G_R^2}{H_R + \lambda} - \frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma \quad (5)$$

G_L (H_L) and G_R (H_R) are the summation of the first (second) derivative of the left and right child, respectively. Different from XGBoost [16], HGB discretizes continuous data instances by placing them into hundreds of bins in advance, making it fast and low memory-consuming.

The goal of Phase I is to eliminate false candidates as many as possible. However, the extreme imbalance between positive and negative candidates (>1:1000 according to industrial collaborator) is prone to a biased selection towards false candidates. Hence, a threshold (denoted by θ_T) is used to adjust for such bias. θ_T is the probability that a candidate is predicted as true. Any candidate, even predicted as a true one but with the probability lower than θ_T, is classified as false and will not be further considered in Phase II. On the other hand, if any true candidate is removed early in Phase I, optimizing Eq. (1) in Phase II only leads to a suboptimal solution at best, as the original global optimum no longer exists. To deal with the problem, we tune the threshold θ_T using the recall metric accordingly. In statistics, *recall*= TP/(TP+FN), TP=true positive, FN = false negative. Experiments show that when recall $\geq$0.9, GEM can take the advantage of the conservative modeling in Phase I and eventually performs well.

C. Phase II: Binary Differential Evolution

Phase II in GEM is developed from a variant of differential evolution algorithm called sabDE [18]. Assuming there are N individuals in the swarm and M candidates, each individual $S_i (i = 1, 2, 3...N)$ represents a solution, which is a M bits binary array and $S_{ij} = 1$ means the jth candidate is present while $S_{ij} = 0$ means the jth candidate is not present. Algorithm 1 evaluates the fitness of S_i.

Algorithm 1 Calculate fitness of S

Input: a solution S; failing patterns FP; failing responses FR; passing patterns PP; passing responses PR;
Output: $fitness$
  $fitness \leftarrow 0$
  for $a = 1$ **to** $|FP|$ **do**
    if $simulate(S, FP_a) = FR_a$ **then**
      $fitness \leftarrow fitness + 2 * (|PP| + 1)$
    end if
  end for
  for $b = 1$ **to** $|PP|$ **do**
    if $simulate(S, PP_b) \neq PR_b$ **then**
      $fitness \leftarrow fitness - |FP|$
    end if
  end for
  $fitness \leftarrow fitness - \varepsilon * CountOne(S)$
  return $fitness$

In Algorithm 1, $|PP|$ and $|FP|$ are the number of passing patterns and failing patterns, respectively. We reward passing patterns twice in the fitness function by $2 * (|PP| + 1)$. ε is a tuning parameter that regulates the selection of solutions for fewer candidates, thus reducing the impact on $fitness$. Phase II stops once a maximum number of iterations is attained or the fitness value ceases to improve for the past certain number of generations. All feasible solutions are recorded once the stopping criterion has been met.

Altogether there are six search strategies, shown in Eq. (6)-(11). An individual in each generation randomly picks one of the six during the exploration of the optimal solution.

$$v_i = S_{r1} \otimes (S_{r2} \oplus S_{r3}) \tag{6}$$
$$v_i = S_{gbest} \otimes (S_{r1} \oplus S_{r2}) \tag{7}$$
$$v_i = S_{r1} \otimes ((S_{r2} \oplus S_{r3}) + (S_{r4} \oplus S_{r5})) \tag{8}$$
$$v_i = S_{gbest} \otimes ((S_{r1} \oplus S_{r2}) + (S_{r3} \oplus S_{r4})) \tag{9}$$
$$v_i = S_i \otimes ((S_{gbest} \oplus S_i) + (S_{r1} \oplus S_{r2})) \tag{10}$$
$$v_i = S_i \otimes ((S_{gbest} \oplus S_i) + (S_{r1} \oplus S_{r2}) + (S_{r3} \oplus S_{r4})) \tag{11}$$

v_i is a temporary variable for the individual S_i. S_{gbest} is the global best individual. S_{r1}, S_{r2}, ... are the random individuals. The operator $\otimes$ performs Exclusive-OR operation. $+$ is an OR operator. The operator $\oplus$ first performs Exclusive-OR operation then heuristically selects a number of bits to reset them to zero. Eq.(6) and (8) are expected to be efficient at exploring the potential solution space. Eq.(7), (9)-(11) start from the best individuals so far to search for better ones. The obtained v_i then undergoes a $crossover$ procedure with S_i to generate a new individual u_i, which is later evaluated for $fitness$ in the evolution iteration. v_i, u_i, and S_i are all vectors of size $m \times 1$, where m is defined in Section II-A as the number of candidates.

Previously, in multiple-fault diagnosis by particle swarm optimization (PSO) [10], an individual only learns either from the global best individual or the best solution it attains so far, following Eq. (10). Comparatively, GEM has six search strategies by Eq. (6)-(11). Variety creates opportunity for better evolution results. Experiments in Section III also proved that GEM outperforms PSO [10].

Phase II in GEM is developed from sabDE [18], with major improvements in the following two steps during the evolution,

1) initialization: Unlike sabDE using a Bernoulli process with the probability set at 0.5, we initialize the swarm population by randomly selecting candidates from the interval $[1, m]$. Therefore, GEM is able to consider the cases when the number of candidates are fewer than $0.5 * N * m$. However, according to differential evolution theory [18], if one candidate fails to survive in the swarm after initialization or an iteration, it will never appear in the solution, leading to the loss of true candidates. On the contrary, if one false candidate is selected by all the individuals in any generation, it will not disappear and thus trap into local optimum. The next modification can relieve the problem.

2) crossover: After the generation of a vector v_i for S_i, $crossover$ randomly chooses a bit from v_i and S_i to generate another vector u_i for evaluation. Before the termination of $crossover$, each u_i has a small chance to set a random bit to one or flip it to zero. By setting bits to one, GEM introduces new candidates into consideration and avoids inadequate search. By flipping bits to zero, GEM drops redundant candidates and thus improves resolution.

III. Experiment

Experiments were performed on eleven circuit designs: three ISCAS'89 benchmark circuits (35932, s38417 and s38584), four ITC'99 circuits (b14, b17, b20 and b22), a level-two cache write-back buffer (L2B) from the *OpenSPARC T2* processor [19], a branch target buffer (BTB) and a float-point-to-integer converter (FPtoINT) from the *Rocket* core [20], and one divider (DIV) component from the *BOOM* core [21]. Full scan chains are implemented to enable sequential circuit testing. For each design, Table II gives the number of test patterns generated by a commercial ATPG tool and its fault coverage. For each fault multiplicity, the average numbers of candidates (rounded to integers) are also provided via fault simulation and path-tracing analytics [15].

TABLE II
CIRCUITS FOR EXPERIMENTS AND INITIAL NUMBER OF CANDIDATES

circuits	#test	coverage	2 faults	5 faults	7 faults	10 faults
s35932	25	90.23%	90	226	313	424
s38417	102	99.73%	430	1032	1419	1987
s38584	152	95.31%	148	332	457	618
b14	733	99.46%	1832	3447	4224	5126
b17	570	98.26%	5656	11834	15434	18374
b20	425	98.45%	4151	7655	9018	10476
b22	434	98.44%	4240	8645	11296	13679
L2B	106	99.96%	202	494	627	860
BTB	1125	100.00%	1822	3856	5040	6179
FPtoINT	445	99.93%	2807	4710	5347	6105
DIV	393	100.00%	3429	5857	6564	7488

To simulate a failing chip, every time up to ten multiple faults are injected into the benchmark design to create the failing population. For each design, 1000 fail logs with ten stuck-at faults injected are used as the training dataset, with an addition of 200 fail logs for validation and parameter tuning; 500 fail logs are generated for testing GEM's performance. All of the multiple faults injected are different. According to [11] [10], the multiplicity of faults examined is up to ten, as a failing chip containing > 10 defects rarely happens.

Table III compares the results of Phase I by histogram gradient boosting (HGB) and random forests (RF). $real$ is the average number of remaining true candidates. $cand$ is the average number of all candidates (including both true and false ones). The number of base tree predictors is 100 for both HGB and RF. The learning rate is set at 0.1 for HGB. The minimum number of samples per leaf is set 20 for HGB and one for RF to allow a tree to grow to its maximum depth.

HGB in Phase I removes most of the false candidates, comparing with the initial number of faults reported in Table II. Table III shows that the average number of remaining candidates is 82.86 by HGB and 119.44 by RF, indicating that HGB removes 98.21% candidates whereas RF removes 97.42%. Meanwhile, HGB keeps 95.22% true candidates, better than RF which keeps 93.11% of the true ones. Comparing with RF, HGB removes 1.31× more false candidates, with an increase of 2.27% true candidates retained. For the purpose of eliminating false candidates but retaining true ones simultaneously, HGB often outperforms RF.

TABLE III
RESULTS FROM GEM PHASE-I (HISTOGRAM GRADIENT BOOSTING, HGB) AND RANDOM FORESTS (RF)

	2 faults				5 faults				7 faults				10 faults			
	real		cand		real		cand		real		cand		real		cand	
circuits	HGB	RF	HGB	RF	HGB	RF	HGB	RF	HGB	RF	HGB	RF	HGB	RF	HGB	RF
s35932	**1.988**	1.972	6.778	**6.620**	**4.910**	4.720	10.638	**10.110**	**6.872**	6.600	14.302	**13.572**	**9.600**	9.074	19.224	**17.810**
s38417	**1.972**	1.964	6.568	**6.006**	**4.820**	4.796	**8.888**	8.698	**6.568**	6.540	**11.486**	11.506	9.014	**9.022**	**14.902**	15.090
s38584	**1.988**	1.936	4.738	**4.408**	**4.862**	4.810	7.530	**7.394**	**6.680**	6.640	9.906	**9.832**	**9.014**	9.012	**12.912**	13.010
b14	**1.982**	1.952	**119.014**	134.174	**4.800**	4.630	**156.512**	158.574	**6.532**	6.244	**172.546**	173.146	**9.092**	8.550	**186.218**	208.172
b17	**1.978**	1.960	**36.194**	100.214	**4.784**	4.726	**57.996**	120.86	**6.588**	6.500	**74.930**	130.420	**9.102**	8.974	**79.000**	157.868
b20	**1.982**	1.950	**366.388**	372.436	**4.820**	4.544	**269.160**	330.690	**6.538**	6.084	**227.184**	301.326	**9.050**	8.320	**179.878**	270.848
b22	**1.978**	1.956	**174.450**	228.078	**4.800**	4.662	**120.748**	214.894	**6.538**	6.236	**105.380**	217.680	**9.034**	8.668	**100.002**	224.714
L2B	**1.992**	1.936	4.842	**3.836**	**4.826**	4.712	7.270	**6.774**	**6.664**	6.572	9.182	**8.908**	**9.054**	9.014	12.364	**11.472**
BTB	**1.952**	1.910	**17.398**	18.252	**4.714**	4.614	**27.930**	34.064	**6.526**	6.464	**26.904**	28.858	9.026	**9.046**	**33.334**	34.876
FPtoINT	**1.994**	1.954	**122.260**	214.302	**4.866**	4.662	**161.822**	212.610	**6.576**	6.316	**192.118**	222.782	**9.008**	8.594	**215.698**	254.116
DIV	**1.980**	1.968	**44.200**	283.540	4.816	**4.818**	**67.316**	159.744	6.664	**6.688**	**72.190**	138.684	9.070	**9.264**	**77.718**	154.530

TABLE IV
DIAGNOSIS QUALITY COMPARISON FOR GEM, PSO [10], AND A COMMERCIAL DIAGNOSTIC TOOL

	2 faults						5 faults					
	diagnosability			resolution			diagnosability			resolution		
circuits	GEM	PSO	Tool	GEM	PSO	Tool	GEM	PSO	Tool	GEM	PSO	Tool
s35932	**0.997**	0.805	0.737	0.567	**0.741**	0.520	**0.981**	0.702	0.716	0.577	**0.667**	0.507
s38417	**0.988**	0.825	0.932	0.753	**0.761**	0.736	**0.966**	0.526	0.910	**0.826**	0.613	0.753
s38584	**0.994**	0.922	0.828	0.871	**0.900**	0.830	**0.974**	0.805	0.805	**0.858**	0.808	0.808
b14	**0.975**	0.464	0.866	**0.807**	0.379	0.475	**0.940**	0.250	0.226	**0.795**	0.261	0.272
b17	**0.987**	0.334	0.801	**0.850**	0.330	0.732	**0.949**	0.110	0.455	**0.838**	0.222	0.552
b20	**0.969**	0.394	0.788	**0.793**	0.282	0.456	**0.931**	0.175	0.209	**0.825**	0.198	0.174
b22	**0.980**	0.391	0.831	**0.767**	0.305	0.616	**0.924**	0.167	0.269	**0.824**	0.216	0.231
L2B	**0.997**	0.929	0.935	**0.951**	0.895	0.941	**0.968**	0.703	0.891	**0.959**	0.779	0.891
BTB	**0.972**	0.639	0.715	**0.896**	0.597	0.884	**0.940**	0.282	0.373	**0.898**	0.427	0.630
FPtoINT	**0.992**	0.528	0.686	**0.818**	0.467	0.608	**0.967**	0.251	0.188	**0.854**	0.372	0.281
DIV	**0.982**	0.459	0.792	**0.885**	0.436	0.843	**0.964**	0.226	0.188	**0.893**	0.381	0.483
	7 faults						10 faults					
	diagnosability			resolution			diagnosability			resolution		
circuits	GEM	PSO	Tool	GEM	PSO	Tool	GEM	PSO	Tool	GEM	PSO	Tool
s35932	**0.982**	0.586	0.635	0.572	**0.606**	0.494	**0.958**	0.398	0.427	**0.569**	0.547	0.519
s38417	**0.942**	0.397	0.867	**0.816**	0.545	0.723	**0.908**	0.241	0.796	**0.835**	0.480	0.736
s38584	**0.955**	0.587	0.784	**0.858**	0.681	0.799	**0.903**	0.453	0.739	**0.860**	0.631	0.799
b14	**0.906**	0.207	0.143	**0.755**	0.250	0.306	**0.877**	0.156	0.098	**0.774**	0.239	0.423
b17	**0.929**	0.087	0.308	**0.832**	0.243	0.472	**0.895**	0.062	0.165	**0.836**	0.239	0.422
b20	**0.903**	0.131	0.137	**0.931**	0.189	0.326	**0.885**	0.101	0.085	**0.828**	0.208	0.503
b22	**0.922**	0.113	0.174	**0.817**	0.183	0.379	**0.893**	0.079	0.104	**0.811**	0.175	0.476
L2B	**0.954**	0.563	0.882	**0.960**	0.707	0.917	**0.908**	0.407	0.837	**0.967**	0.658	0.883
BTB	**0.931**	0.199	0.278	**0.901**	0.432	0.697	**0.902**	0.128	0.175	**0.901**	0.388	0.620
FPtoINT	**0.931**	0.186	0.125	**0.875**	0.346	0.373	**0.892**	0.139	0.080	**0.849**	0.335	0.426
DIV	**0.942**	0.183	0.122	**0.880**	0.390	0.547	**0.902**	0.125	0.084	**0.891**	0.373	0.610

Table IV compares the diagnostic results by GEM, PSO [10], and a state-of-the-art commercial diagnosis tool (denoted by Tool). For GEM, the number of individuals is 50. Maximum number of iterations is five times the number of possible faults. If the fitness function does not improve for the past 150 consecutive iterations, evolution stops. Because PSO has no pre-filtering step like Phase I in GEM, the number of individuals and the maximum number of iterations are increased to 500 and 100, respectively. $diagnosability$ is the average ratio of the number of true candidates to the number of injected faults. $resolution$ is the average ratio of the number of true candidates to the number of reported candidates.

For 2-, 5-, 7, 10-multiple faults, the average diagnosability of GEM is 98.48%, 95.49%, 93.61%, and 90.21%, respectively, with the average resolution being 81.44%, 83.15%, 83.61%, and 82.92%, respectively. The average diagnosability of Tool is 81.01% (1.22×)[1], 47.55% (2.01×), 40.50% (2.31×), and 32.64% (2.76×), respectively; the resolution of Tool is 69.46% (1.17×), 50.75% (1.64×), 54.85% (1.34×), and 58.34% (1.42×). The average diagnosability of PSO is 60.82% (1.62×), 38.15% (2.50×), 29.28% (3.20×), and 20.81% (4.33×); the resolution of PSO is 55.39% (1.47×), 44.95% (1.85×), 41.56% (2.01×), and 38.85% (2.13×), respectively. GEM clearly outperforms both Tool and PSO. In addition, it can be observed that as the multiplicity grows from two to ten, the diagnosability of Tool and PSO deteriorates rapidly but that of GEM decreases slightly.

We assess the effectiveness of GEM, PSO, and Tool in estimating the actual number of injected faults. Fig. 2 illustrates the distribution of estimated multiplicity for an example design L2B by GEM, PSO, and Tool, respectively. As the multiplicity increases, their performance decreases. The distributions of GEM and Tool are more tightly clustered for 2-, 5-, 7-multiple faults, hence showing no boxes for those three cases. GEM is as good as Tool for estimating the possible fault quantity, better than PSO that often underestimates these numbers.

[1]The numbers in (·) are the times GEM outperforms the compared method.

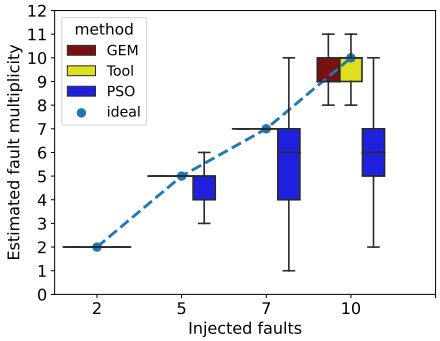

Fig. 2. Box-plot of multiplicity estimation by GEM, PSO, and Tool.

Fig. 3 shows the runtime. For the runtime spent in each main process carried out in GEM, path-tracing takes 3.38%, feature extraction takes 6.62%, Phase I HGB prediction consumes 0.25%, Phase II takes 89.75%. On average, the commercial Tool takes 2.41s to finish execution, which is the fastest among three. PSO takes 129.81s for diagnosis while GEM uses 19.91s, 6.52× faster than PSO. The acceleration in GEM can be attributed to the reduction in the search space of possible fault combinations during Phase I, enabling a faster convergence for evolution computation in Phase II.

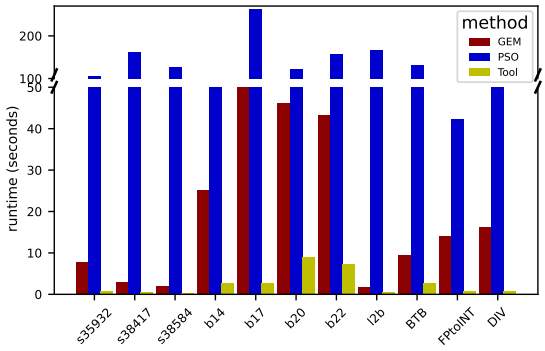

Fig. 3. Runtime comparison of GEM, PSO, and Tool.

IV. Conclusions

Multiple-faults diagnosis is intrinsically difficult. Typically, all possible fault combinations need to be examined hoping for the best solution. Unfortunately, it is either infeasible or computationally challenged.

GEM is a hybrid multiple-fault diagnosis system that takes the advantage of both ML and combinatorial optimization. Phase I in GEM uses gradient boosting for fast modeling the stuck-at fault simulation and X-fault simulation responses to filter out a significant amount of false candidates, while retaining correct ones. Phase II is developed from a differential evolution algorithm, with major adaptations fitting into the diagnosis scenario. It iteratively search the remaining possible fault combinations from Phase I for the best possible one. The introduced randomness (shown in Eq. (6)-(11)) makes the evolution computation more efficient to converge.

In experiments, a totally of 35,200 fail logs are generated. On average, Phase I removes 98.21% candidates with 95.22% true candidates retained. Phase II achieves the diagnosability of 94.45% and resolution of 82.78%, respectively. GEM outperforms an heuristic-based work [10], a ML-based one [11], and a state-of-the-art commercial diagnostic tool, showing its promising for multiple-fault diagnosis.

References

[1] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI test principles and architectures: design for testability*. Elsevier, 2006.

[2] V. Boppana and M. Fujita, "Modeling the unknown! towards model-independent fault and error diagnosis," in *Proceedings IEEE International Test Conference*, 1998, pp. 1094–1101.

[3] J. Ye, Y. Hu, and X. Li, "Diagnosis of multiple arbitrary faults with mask and reinforcement effect," in *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010, pp. 885–890.

[4] I. Pomeranz, V. Chickermane, and S. Venkataraman, "Observation point placement for improved logic diagnosis based on large sets of candidate faults," in *IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–6.

[5] P. Wang, A. M. Gharehbaghi, and M. Fujita, "An incremental automatic test pattern generation method for multiple stuck-at faults," in *IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–6.

[6] X. Wen, T. Miyoshi, S. Kajihara, L.-T. Wang, K. Saluja, and K. Kinoshita, "On per-test fault diagnosis using the x-fault model," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2004, pp. 633–640.

[7] X. Yu and R. D. Blanton, "Diagnosis of integrated circuits with multiple defects of arbitrary characteristics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 6, pp. 977–987, 2010.

[8] J. Ye, Y. Hu, X. Li, W.-T. Cheng, Y. Huang, and H. Tang, "Diagnose failures caused by multiple locations at a time," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, no. 4, pp. 824–837, 2014.

[9] Y.-S. You, C.-Y. Liu, M.-T. Wu, P.-W. Chen, and J. C.-M. Li, "Diagnosis technique for clustered multiple transition delay faults," in *IEEE International Test Conference in Asia (ITC-Asia)*, 2020, pp. 53–58.

[10] S. Kundu, A. Jha, S. Chattopadhyay, I. Sengupta, and R. Kapur, "Framework for multiple-fault diagnosis based on multiple fault simulation using particle swarm optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 696–700, 2014.

[11] S. Mittal and R. D. Blanton, "A deterministic-statistical multiple-defect diagnosis methodology," in *IEEE 38th VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[13] T. H. Kim, H. Lim, M. Cheong, H. Yun, and S. Kang, "Logic diagnosis based on deep learning for multiple faults," in *19th International SoC Design Conference (ISOCC)*, 2022, pp. 366–367.

[14] D. Golovin *et al.*, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017, pp. 1487–1495.

[15] S. Venkataraman and W. K. Fuchs, "A deductive technique for diagnosis of bridging faults," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, 1997, pp. 562–567.

[16] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 785–794.

[17] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] A. Banitalebi, M. I. A. Aziz, and Z. A. Aziz, "A self-adaptive binary differential evolution algorithm for large scale binary optimization problems," *Information Sciences*, vol. 367/368, pp. 487–511, 2016.

[19] I. Parulkar *et al.*, "Opensparc: An open platform for hardware reliability experimentation," in *Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE)*. Citeseer, 2008, pp. 1–6.

[20] K. Asanovic *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[21] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom SOCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.