Prioritizing Tests for Fault Localization through Ambiguity Group Reduction

Alberto Gonzalez-Sanchez[1] Rui Abreu[2] Hans-Gerhard Gross[1] Arjan J.C. van Gemund[1]

[1]*Delft University of Technology, Software Technology Department*
Mekelweg 4, 2628 CD Delft, The Netherlands
Email: {a.gonzalezsanchez,h.g.gross,a.j.c.vangemund}@tudelft.nl

[2]*University of Porto, Departament of Informatics Engineering*
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
Email: rui@computer.org

***Abstract*—In practically all development processes, regression tests are used to detect the presence of faults after a modification. If faults are detected, a fault localization algorithm can be used to reduce the manual inspection cost. However, while using test case prioritization to enhance the rate of fault detection of the test suite (e.g., statement coverage), the diagnostic information gain per test is not optimal, which results in needless inspection cost during diagnosis.**

We present RAPTOR, a test prioritization algorithm for fault localization, based on reducing the similarity between statement execution patterns as the testing progresses. Unlike previous diagnostic prioritization algorithms, RAPTOR does not require false negative information, and is much less complex. Experimental results from the Software Infrastructure Repository's benchmarks show that RAPTOR is the best technique under realistic conditions, with average cost reductions of 40% with respect to the next best technique, with negligible impact on fault detection capability.

I. INTRODUCTION

Regression testing is a time-consuming but rather important task for improving software reliability after a software change. Literature shows how source code changes are bound to introduce regressions in up to a 70% of cases [27]. Given the significant cost associated with regression tests, *test prioritization* has emerged as a predominant technique to reduce testing cost. Prioritized regression test suites aim at detecting failures as soon as possible in order to reduce testing effort (C_t) [14], [15], [21], [24], [28], [35], [38], [43]. The sooner failures are found, the sooner debugging can commence. Automatic fault localization techniques [4], [26], [3], [42], [45], [29] use the information obtained during regression testing (e.g., test coverage and failure information) to produce a ranking of source code statements likely to be the root cause of the observed failures. This ranking is used to minimize the diagnostic work the developer has to perform when inspecting the program to find the faults ("inspection cost", C_d) [1], [4], [26].

Previous work in the combination of test prioritization and diagnosis [25], [44] has shown that, while traditional prioritization techniques minimize C_t by merely reducing the delay between testing and debugging, they do not maximize diagnostic information, and therefore increase inspection effort C_d. The reason is that traditional test prioritization aims at high code coverage, whereas obtaining a refined diagnosis requires partially *revisiting* already covered code to further exonerate or indict suspect statements. In order to minimize overall cost, one must consider a test prioritization strategy that takes into account the combined cost of testing and debugging, $C_t + C_d$[1].

Recently, a *diagnostic test prioritization* technique (i.e., test prioritization technique that maximizes the diagnostic information gain per test) has been proposed [17], [19] to solve this problem. The current approach to diagnostic test prioritization is based on the *information gain* heuristic (IG). Although in theory its diagnostic effectiveness is optimal [17], it does have a number of drawbacks that restrict its applicability in practice. First, its performance depends heavily on the precise estimation of a number of parameters (fault density and false negative test rate) that are very expensive to obtain, as well as very error-prone. This has been shown to severely impact on the quality of the IG heuristic [17], [19]. Second, the algorithmic complexity of IG is exponential, due to the fact that more than one fault may be present in the system. Finally, unlike traditional test prioritization techniques, in IG-based diagnostic test prioritization the tests are selected on-line, based on the actual pass/fail results of previously executed tests (which varies per regression cycle). On-line calculations impose a large overhead in the test process if test cases are relatively short.

In this paper we introduce a novel, off-line diagnostic test prioritization algorithm that solves the above problems. In particular, our paper makes the following contributions:

- We present RAPTOR (gReedy diAgnostic Prioritization by ambiguiTy grOup Reduction), a low-complexity diagnostic test prioritization algorithm that can be used

[1]The addition is not necessarily arithmetic as inspection cost typically involves labor by the diagnostician

off-line, which does not require any additional input parameters, and can consider variance in test costs.
- We evaluate the performance of RAPTOR for the Siemens set as well as for 5 larger programs available from the Software Infrastructure Repository (SIR) [12]. All programs are extended to accommodate multiple faults.

In the above experiments we compare the performance of RAPTOR to the contemporary test prioritization algorithms ADDST [35], FEP [35], ART [24], the IG-based diagnostic test prioritization algorithm SEQUOIA [17], and to random prioritization (RND), our baseline.

To the best of our knowledge, a diagnostic approach to off-line test prioritization has not been described before. Our results show that RAPTOR can deliver a performance, in terms of inspection cost reduction per unit of test cost, superior to ADDST, FEP, ART, and RND, and, in some cases, even to the theoretical optimum, SEQUOIA. Improvements are, on average, 40% better with respect to the next best technique, and 55% better than the baseline (RND).

The paper is organized as follows. In the next section we present the basic idea behind diagnostic prioritization. In Section III we present related prioritization and diagnosis techniques. In Section IV RAPTOR is presented. In Section V we present experimental performance results. Section VI discusses these results and the threats to their validity. Related work is discussed in Section VII. In Section VIII, we conclude and discuss future work.

II. DIAGNOSTIC PRIORITIZATION

Before going into the specific details of our diagnostic prioritization algorithm, we first explain why traditional test prioritization conflicts with the goal of fault localization. Let us consider the faulty program in Table I. We provide a test suite with 8 tests that provide full statement coverage. The following inputs are needed for test prioritization and diagnosis:

- A finite set $\mathcal{C} = \{c_1, \ldots, c_j, \ldots, c_M\}$ of M components (typically source code statements) of which $M_f = 0, \ldots, M$ can be faulty.
- A finite set $\mathcal{T} = \{t_1, \ldots, t_i, \ldots, t_N\}$ of N tests with binary outcomes o_i, where $o_i = 1$ if test t_i failed, and $o_i = 0$ otherwise. Each test has a cost $C_t(i)$.
- A $N \times M$ coverage matrix, $A = [a_{ij}]$, where $a_{ij} = 1$ if test t_i involves component c_j, and 0 otherwise. Each row is called "spectrum", and each column is called "signature".

These inputs are represented in the example program of Table I. For the sake of code readability, the coverage matrix is shown transposed. For exposition clarity, we use a simplified way of diagnosis. We assume that, if a test fails, the statements that were covered become suspects, and if it passes, the covered statements are exonerated.

Diagnostic performance is expressed in terms of a cost metric C_d that measures the excess *effort* incurred in finding all statements at fault (inspection effort) [4]. C_d measures *wasted* effort, independent of the number of faults M_f in the program,

		$\mathcal{T}$							
$\mathcal{C}$	Program: Character Counter	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
	`function count(char * s) {`								
	`int let, dig, other, i;`	A (transposed)							
c_1	`while(c = s[i++]) {`	1	1	1	1	1	1	1	1
c_2	`if ('A'<=c && 'Z'>=c)`	1	1	1	1	1	1	0	1
c_3	**`let += 2; // FAULT`**	1	1	1	1	1	1	0	0
c_4	`elsif ('a'<=c && 'z'>=c)`	1	1	1	1	1	0	0	1
c_5	`let += 1;`	1	1	0	0	1	0	0	0
c_6	`elsif ('0'<=c && '9'>=c)`	1	1	1	1	0	0	0	1
c_7	`dig += 1;`	0	1	0	1	0	0	0	0
c_8	`elsif (isprint(c))`	1	0	1	0	0	0	0	1
c_9	`other += 1;`	1	0	1	0	0	0	0	1
c_{10}	`printf("%d %d %d\n",`	1	1	1	1	1	1	1	1
	`let, dig, other);}`								
	Test case outcomes	F	F	F	F	F	F	P	P

TABLE I
EXAMPLE PROGRAM AND INPUTS FOR TEST PRIORITIZATION AND DIAGNOSIS

to enable an unbiased evaluation of the effect of M_f on C_d. Thus, regardless of M_f, $C_d = 0$ represents an ideal diagnosis technique (all M_f faulty statements on top of the ranking, no effort wasted on inspecting other statements to find they are not faulty), while $C_d = M - M_f$ represents the worst case (inspecting all $M - M_f$ healthy statements until arriving at the M_f faulty ones).

A. Example

A traditional prioritization heuristic, such as the additional statement coverage heuristic (ADDST) [35] tries to maximize covered code in as few tests as possible. Once all the code is covered, the algorithm restarts with the remaining tests. Such algorithm would start by selecting t_1, which fails. If we tried to apply diagnosis after this test, a large number of source code statements would have to be inspected, since all but statement c_7 are covered. Assuming random inspection, $C_d = (9 - 1)/2 = 4$ statements in average. If we would execute one more test, ADDST would select t_2. Since it also fails, c_8 and c_9 are no longer suspects and inspection effort would drop to $C_d = (7 - 1)/2 = 3$ on average. After t_3, c_5 is also exonerated and $C_d = (6 - 1)/2 = 2.5$ on average. Executing the next two ADDST tests, t_4 and t_5, would not provide any new diagnostic information, since they cover all the current suspects. Executing t_8 as the last test (pass) would finally enable a perfect diagnosis since the only non-covered suspect is c_3.

On the other hand, RAPTOR, the diagnostic prioritization algorithm presented in this paper would initially select t_5 since it provides a better balance between covered and non-covered statements. As it fails, only the 6 covered statements become suspects. This means that, after just one test, $C_d = 2.5$, already equal to ADDST after 3 tests. RAPTOR tries to increase the difference between the current suspects, and chooses t_8 as second test, which passes. The set of suspects is reduced to only c_3 and c_5, and $C_d = 0.5$. Test t_6 is selected next, to differentiate between those, and fails, indicating that the fault must be c_3. The remaining tests do not change the result.

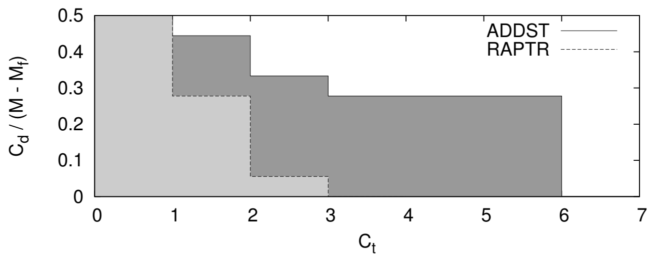

Fig. 1. Advantage of diagnostic prioritization

B. Measurement

Whereas both algorithms detect the presence of a fault after the first test, RAPTOR produces much more diagnostic information per test, and the overall cost combination $C_t + C_d$ is significantly reduced. To measure how good a specific prioritization algorithm is for fault localization, we need to measure the evolution of C_d per test case choice. Previous work [25], [44] measured the relative difference between C_d at fixed percentages of the test suite and the C_d after applying 100% of the test suite, following the formula

$$RelativeExpense(n\%) = \frac{C_d(n\%) - C_d(100\%)}{C_d(100\%)}$$

In [25], the first sample is taken after a 10% of the test cases have been executed, which corresponds to approximately 25 tests in their experimental setting. However, previous experiments [2] have shown that the most gains in inspection cost happen in the first $\sim$ 20 tests. Therefore, we use a continuous measure, instead of discrete samples, to evaluate this continuous process without running the risk of missing important details or biasing the results.

Our *effectiveness* measure can be seen as a continuous variant of $RelativeExpense$, and uses the area difference between $C_d(n)$ (i.e., C_d as a function of the number of applied tests) and $C_d(100\%)$, according to

$$S = \sum_{n=1}^{N} \left(\frac{C_d(n-1) - C_d(N)}{M - M_f} \cdot C_t(n) \right) \tag{1}$$

where $C_t(n)$ corresponds to the cost of the n-th test. A similar measurement is used in [17], [19]. Figure 1 illustrates the evolution of inspection cost C_d per unit of test cost C_t for both scenarios, and the area S (shaded).

The effectiveness of ADDST in our example corresponds to $S = \frac{1}{9} \cdot (4.5 + 4 + 3 + 2.5 + 2.5 + 2.5) = 2.1$, whereas the effectiveness of RAPTOR corresponds to $S = \frac{1}{9} \cdot (4.5 + 2.5 + 0.5) = 0.83$, a 60% cost reduction.

III. Related Techniques

In general terms, testers will apply one of the prioritization algorithms to obtain an ordered test suite. The tests in this ordered suite will be executed one by one recording the test outcomes, until, e.g., the available testing time is exhausted. If failures have occurred, a diagnosis algorithm is used to locate the faults causing the failures, producing a diagnosis. Since we cannot directly measure the diagnostic quality of the information provided by the ordered test suite, we will use two well-known diagnostic techniques and infer conclusions based on the quality of the two different diagnoses they produce. In the following we introduce basic concepts on test prioritization algorithms as well as on the diagnosis techniques considered.

A. Test Prioritization

Traditionally, test cases have been prioritized with the goal of enhancing their fault *detection* ability, i.e., the rate at which failures are produced. Diagnostic test prioritization, a recent development, proposes to prioritize tests with the goal of fault *localization*, i.e., the rate at which diagnostic quality improves.

Random: this is the most straightforward prioritization criterion, which orders test cases according to random permutations of the original test suite. It is used as baseline in many prioritization experiments [15], [35], [38].

Add-Statement: In additional statement coverage prioritization (ADDST) [15], [35], test cases are selected iteratively in terms of the additional coverage they yield, taking into account all the test cases that were already executed. The algorithmic complexity (time and space) of ADDST prioritization is $O(N \cdot M)$ per selected test.

Fault-exposing Potential: FEP is a coverage-based prioritization algorithm that assigns each statement a confidence value [35]. As high confidence is assigned to a statement that has been exercised by a number of (passing) tests, those statements need less coverage in subsequent tests. FEP has $O(M \cdot N)$ time complexity.

Adaptive Random Testing: ART is a hybrid random/coverage-based algorithm [24]. ART selects its next test case in two steps. First, it samples tests randomly until one of the samples does not add additional coverage. Second, it selects the test which maximizes a distance function with the already selected test cases. The best-case time complexity is $O(N^2)$ per selected test, while the worst-case time complexity is $O(N^2 \cdot M)$.

Information Gain: Unlike all the previous techniques, Information gain (IG) is a diagnostic prioritization technique. IG orders test cases dynamically (on-line), based on the improvement of diagnostic quality, based on an information-theoretical measurement of the quality of the current diagnosis D, the probability of the test passing or failing, and the quality of the diagnosis if the test passes or fails. IG is based on information theory, in terms of *entropy* [11]. The complexity of IG is $O(N \cdot 2^M)$ for multiple faults. An approximate solution with near-optimal performance, dubbed SEQUOIA, can perform IG test selections in $O(M \cdot N)$ time [17].

B. Fault Localization

Although there is a large number of different fault localization techniques (see Section VII), in our work we will consider two well-known coverage-based techniques: Bayesian diagnosis (well-known from Model-Based Diagnosis, an area within AI), and statistical approaches such as Tarantula [26], and Ochiai [3].

Statistical Fault Diagnosis: In statistical fault diagnosis, the likelihood a component is at fault is quantified in terms of *similarity coefficients* (SCs). A SC measures the statistical similarity between statement c_j's test coverage $(a_{1j}, \ldots, a_{Nj})$ and the observed test outcomes, $(o_1, \ldots, o_N)$. Similarity is computed by means of four counters $n_{pq}(j)$ that count the times a_{ij} and o_i form the combinations $(0,0), \ldots, (1,1)$, respectively, i.e.,

$$n_{pq}(j) = |\{i \mid a_{ij} = p \wedge o_i = q\}| \;\; p, q \in \{0,1\} \tag{2}$$

For instance, $n_{10}(j)$ and $n_{11}(j)$ are the number of tests in which c_j is executed, and which passed or failed, respectively. The four counters sum up to the number of tests N. In this paper we will consider the Ochiai SC, given by

$$\text{Ochiai:} \quad l_j = \frac{n_{11}(j)}{\sqrt{(n_{11}(j)+n_{01}(j))\cdot(n_{11}(j)+n_{10}(j))}} \tag{3}$$

A great advantage of SCs is their ultra-low computational complexity compared to probabilistic approaches. Despite their lower diagnostic accuracy [4] SCs have, therefore, gained much interest.

In our example system, after executing all the tests in the test suite, the counters for c_3 are $n_{11}(3) = 6$, $n_{10}(3) = 0$, $n_{01}(3) = 0$, $n_{00}(3) = 2$. Its likelihood being the faulty one according to Ochiai is $l_3 = 1.0$. The remaining statements all have lower likelihoods, as they all have $n_{10}(j) > 0$ or $n_{01}(j) > 0$. For example, for c_2, $n_{11}(2) = 6$, $n_{10}(2) = 1$, $n_{01}(2) = 0$, $n_{00}(2) = 1$, and its Ochiai similarity is $l_2 = 0.92$.

Bayesian Diagnosis: Bayesian diagnosis is an alternative diagnostic technique founded on probability theory, aimed at obtaining a set of fault *candidates* $D = \langle d_1, \ldots, d_k \rangle$. Each candidate d_k is a subset of the statements which, when simultaneously at fault, explain the observed failures. For instance, $d = \{c_1, c_3, c_4\}$ indicates that c_1 **and** c_3 **and** c_4 are faulty, and no other statement. Many algorithms exist to compute D [10], [31].

The candidates returned by Bayesian fault diagnosis are ordered according to their probability of being the true diagnosis $\Pr(d_k)$. Initially, the probability of each candidate depends on the estimated probability of each of its statements being faulty, (p_j, prior fault probability). After test case t_i is executed, the probability of each candidate is updated depending on the outcome o_i of the test, following Bayes' rule:

$$\Pr(d_k|o_i) = \frac{\Pr(o_i|d_k)}{\Pr(o_i)} \cdot \Pr(d_k) \tag{4}$$

In this equation, $\Pr(d_k)$ represents the prior probability of candidate d_k before the test is executed. $\Pr(o_i)$ is a normalization value that represents the residual probability of the observed outcome, and $\Pr(o_i|d_k)$ represents the probability of the observed outcome o_i produced by a test t_i, if that candidate d_k was the actual diagnosis. This depends on the false negative rate (FNR), of the statements in d_k. In software fault localization, FNR is related to the concepts of testability [40], and coincidental correctness [41]. A more detailed description of Bayesian diagnosis can be found in [11], [4], [16].

To diagnose the example system in Table I, for simplicity we will assume here that all statements have equal prior probability $p_j = 0.1$ (software has a much lower prior probability, this is just an example), and we assume 0% FNR, i.e., faults will always cause a failure. The initial probability for $d_3 = \{c_3\}$, is $\Pr(d_3) = 0.1^1 \cdot 0.9^9 = 0.038$. A more complex candidate such as $d_{3,5} = \{c_3, c_5\}$ has an initial probability equal to $\Pr(d_{3,5}) = 0.1^2 \cdot 0.9^8 = 0.004$. After all tests are executed, the diagnosis is composed by 4 candidates $\Pr(d_3|o_1, o_2, \ldots) = 0.89$, $\Pr(d_{3,5}|o_1, o_2, \ldots) = 0.09$, $\Pr(d_{3,7}|o_1, o_2, \ldots) = 0.09$, and $\Pr(d_{3,5,7}|o_1, o_2, \ldots) = 0.01$. Hence, $C_d = 0$.

IV. RAPTOR

The IG heuristic for diagnostic prioritization requires very precise estimations of the false negative rate of test cases to be able to predict their failure probability. Estimating FNR is a costly and error-prone calculation. Experiments with real programs showed that errors in the FNR estimations can have disastrous results [17]. The enormous investment in parameter estimation and the overhead incurred in the testing process totally outweighs the benefits of IG diagnostic prioritization. Furthermore, since IG requires knowing the outcome of the previous test case, IG test sequences cannot be pre-calculated off-line, introducing an overhead in the testing process.

For these reasons, we present an alternative, off-line diagnostic prioritization algorithm, RAPTOR (*gReedy diAgnostic Prioritization by ambiguiTy grOup Reduction*), that, while not being theoretically optimal, produces competitive results. Furthermore, since it is not affected by FNR estimation errors, its results are in many cases better than IG.

A. Ambiguity

Since the basic working principle of any coverage-based diagnostic algorithm is analyzing the differences and similarities between the signatures of each of the statements of the system under test (each of the columns in the test coverage matrix A) and the failure pattern of the system (the test outcomes o_i), we can use the maximization of these differences to devise a new, simpler diagnostic test prioritization heuristic.

Any individual statement belonging to a group of statements with identical columns (signatures) cannot be uniquely identified as faulty. Such a group is termed an *ambiguity group* [39]. The example system in Table I has two ambiguity groups: $\{1, 10\}$ and $\{8, 9\}$.

Let $AG = \{g_1, g_2, \ldots, g_L\}$ be the ambiguity groups generated by the matrix A, or a subset of it. The residual diagnostic effort should a fault occur in a given group g_i corresponds to the expected diagnostic effort if the developer would randomly pick statements in g_i for inspection: $E[C_d] = \frac{|g_i|-1}{2}$. To keep the heuristic simple, we assume faults are distributed uniformly through the program's code. Therefore the probability of group g_i occurring is $\Pr(g_i) = \frac{|g_i|}{M}$.

The value of the *diagnostic ambiguity* heuristic can be defined by averaging the effort in each group g_i, by the probability of the system containing a fault belonging to each of the ambiguity groups.

$$\mathrm{G}(AG) = \sum_{i=1}^{L} \frac{|g_i|}{M} \cdot \frac{|g_i| - 1}{2} \tag{5}$$

where M is the total number of statements in the system. The complete matrix in Table I produces the set $AG = \{\{1, 10\}, \{8, 9\}\}$ whose ambiguity is $G(AG) = \frac{2}{10} \cdot \frac{1}{2} + \frac{2}{10} \cdot \frac{1}{2} = 0.2$.

B. Ambiguity Reduction

Since $\mathrm{G}(AG)$ estimates the residual diagnostic effort C_d, it can be seen as an estimation of diagnosis quality, and its value can be used to construct an alternative heuristic to IG. Each test that is executed breaks each ambiguity group into two smaller ambiguity groups, one corresponding to the statements in the ambiguity group that are covered by the test, and one corresponding to the statements that are not covered. Groups of size 1 are discarded. Using $\mathrm{G}(AG)$ has the advantage that it does not require knowing the test outcomes, i.e., can be performed off-line, nor requires estimation of failure probabilities (unlike IG), nor does it need additional input parameters, e.g., FNR.

The *ambiguity reduction* heuristic is defined as the difference in ambiguity caused by appending one more test to the test matrix, according to

$$\mathrm{AR}(A, t_i) = G(AG(A)) - G(AG(A||t_i)) \tag{6}$$

where the function AG returns the set of ambiguity groups of a test matrix, and the $||$ operator appends test t_i to A.

C. Algorithm

Algorithm 1 details the implementation of the RAPTOR algorithm, which computes a sequence that optimizes the evolution of ambiguity in a greedy, incremental way.

Algorithm 1 RAPTOR

```
function RAPTOR($T$)
    $A \leftarrow \emptyset$
    while $T \neq \emptyset$ do
        $t_i \leftarrow \arg\max_{i \in T} \frac{\mathrm{AR}(A, t_i)}{C_t(i)}$
        if $\mathrm{AR}(A, t_i) > 0$ then                ▷ Any improvement?
            $A \leftarrow A||t_i$
            $T \leftarrow T \setminus \{i\}$
        else
            $A' \leftarrow$ RAPTOR($T$)
            return $A||A'$
    return $A$
```

The algorithm receives the set of test cases and their coverage as input, together with the test cost. At every iteration, a test t_i is selected, such that it maximizes the reduction of ambiguity per unit of test cost. The test that produces the highest improvement is appended to the result (operator $||$) and removed from the set of free tests. Since the algorithm may reach a point where no test can produce any improvement, the algorithm checks for this case and, if detected, starts recursively with the remaining tests, and appends the resulting A' to A. This policy is also used in [35]. An alternative policy also proposed in [35] would be to simply randomly order the remaining tests.

$G(AG)$ can be computed in at most $O(M)$ time, and the set of ambiguity groups can be computed incrementally in $O(M)$ time from the previous set. Since these two functions have to be repeatedly used for each of the remaining free tests, the complexity per text choice, i.e., for the loop of RAPTOR equals $O(MN)$, a complexity similar to ADDST prioritization. Execution time measurements confirm our complexity analysis. On an 8-core 2.6 GHz. Xeon machine with 32Gb of memory, test selection for $M = 1000, N = 1000$ test matrices takes less than 1 second.

1) Example: RAPTOR would prioritize the tests in our example test suite as follows.

Initially, there is no distinction between statements, they all belong to the same ambiguity group $\{\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\}$, and $\mathrm{G}(AG) = 4.50$.

If we executed test t_1, the ambiguity group will be broken to $\{\{1, 2, 3, 4, 5, 6, 8, 9, 10\}, \{7\}\}$ ($\{7\}$ is discarded as it is a single statement) with ambiguity $\mathrm{G}(AG) = \frac{9}{10} \cdot \frac{8}{2} = 3.6$ On the other hand, if we execute t_5, we obtain $\{\{1, 2, 3, 4, 5, 10\}, \{6, 7, 8, 9\}\}$ and ambiguity $\mathrm{G}(AG) = \frac{6}{10} \cdot \frac{5}{2} + \frac{4}{10} \cdot \frac{3}{2} = 2.1$ which represents a larger reduction of ambiguity (t_6 also provides the same reduction).

The second test chosen by RAPTOR is t_8. The set $\{1, 2, 3, 4, 5, 10\}$ is split into two ambiguity groups, one for the covered statements, $\{1, 2, 4, 10\}$, and one for the not covered, $\{3, 5\}$. The same applies to the $\{6, 7, 8, 9\}$ group, split in $\{6, 8, 9\}$ and $\{7\}$. This new state has a total ambiguity $\mathrm{G}(AG) = \frac{4}{10} \cdot \frac{3}{2} + \frac{2}{10} \cdot \frac{1}{2} + \frac{3}{10} \cdot \frac{2}{2} = 1.00$

Table II shows the sequence of tests chosen by RAPTOR together with the evolution of the ambiguity groups and the value of diagnostic ambiguity. The '(R)' symbol indicates when the set of ambiguity groups is reset to $\{\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\}$ because no test can further reduce the ambiguity.

V. EMPIRICAL EVALUATION

In this section we compare the performance of RAPTOR to RND, ART, ADDST, FEP, and SEQUOIA. The prioritization performance of each algorithm is assessed by the effectiveness formula in Equation 1. Finally, we also study the trade off between fault detection and fault localization made by RAPTOR.

A. Subject Programs

We perform our experiments using the well-known Siemens benchmark set [22], as well as the `flex`, `grep`, `gzip`, `sed`, and `space` programs (obtained from SIR [12]). Every program has a correct version, and a set of test inputs is also provided, which were created with the intention of providing full test coverage. Table III provides more information about the programs used in the experiments, where M corresponds to the number of lines of code.

Test	o_i	Ambiguity Groups	G(AG)	D	C_d
		$\{1,2,3,4,5,6,7,8,9,10\}$	4.5	$\langle\{\},\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\ldots,\{1,2\},\{1,3\}\ldots\{1,2,3\}\ldots\rangle$	4.5
t_5	1	$\{1,2,3,4,5,10\},\{6,7,8,9\}$	2.1	$\langle\{1\},\{2\},\{3\},\{4\},\{5\},\{10\},\{1,2\},\{1,3\}\ldots\{3,5\}\ldots\{1,2,3\}\ldots\rangle$	2.5
t_8	0	$\{1,2,4,10\},\{3,5\},\{6,8,9\},\{7\}$	1.0	$\langle\{3\},\{5\},\{3,5\},\{3,7\},\{5,7\},\{3,5,7\}\rangle$	0.5
t_6	1	$\{1,2,10\},\{3\},\{4\},\{5\},\{6,8,9\},\{7\}$	0.6	$\langle\{3\},\{3,5\},\{3,7\},\{3,5,7\}\rangle$	0.0
t_7	0	$\{1,10\},\{2\},\{3\},\{4\},\{5\},\{6,8,9\},\{7\}$	0.4	$\langle\{3\},\{3,5\},\{3,7\},\{3,5,7\}\rangle$	0.0
t_2	1	$\{1,10\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8,9\}$	0.2 (R)	$\langle\{3\},\{3,5\},\{3,7\},\{3,5,7\}\rangle$	0.0
t_4	1	$\{1,2,3,4,6,7,10\},\{5,8,9\}$	2.4	$\langle\{3\},\{3,5\},\{3,7\},\{3,5,7\}\rangle$	0.0
t_3	1	$\{1,2,3,4,6,10\},\{7\},\{5\},\{8,9\}$	1.6 (R)	$\langle\{3\},\{3,5\},\{3,7\},\{3,5,7\}\rangle$	0.0
t_1	1	$\{1,2,3,4,5,6,8,9,10\},\{7\}$	1.6	$\langle\{3\},\{3,5\},\{3,7\},\{3,5,7\}\rangle$	0.0

TABLE II
EVOLUTION OF AMBIGUITY GROUPS AND D FOR THE RAPTOR HEURISTIC FOR OUR EXAMPLE SYSTEM

	Program Name	Mutants	LOC(M)	N	Program Type
pt	print_tokens	491	539	4,130	Lexical Analyzer
p2	print_tokens2	294	489	4,115	Lexical Analyzer
re	replace	757	507	5,542	Pattern Recognition
sc	schedule	281	397	2,650	Priority Scheduler
s2	schedule2	212	299	2,710	Priority Scheduler
tc	tcas	208	174	1,608	Altitude Separation
ti	tot_info	396	398	1,052	Information Measure
sp	space	6,283	9,126	500	ADL Parser
gz	gzip	20,811	6708	211	Compressor
se	sed	11,050	9014	184	Stream Editor
gr	grep	17,707	13,287	809	String Matching
fl	flex	15,574	14,194	107	Lexer Generator

TABLE III
PROGRAMS USED FOR EVALUATION

To obtain a realistic fault sample size, we have extended the subject programs with versions (mutants) for which we seeded random combinations of *multiple* faults similar to the ones in Siemens. These additional faults are obtained by applying mutation techniques [5], [32] to the reference implementation of each program. The mutants were generated by a modified version of `Zoltar`, a spectrum-based fault localization tool set [23]. As each program includes a correct version, we use the output of the correct version as oracle. For each program, 100 faulty versions seeded with 1 and 3 random mutation faults were created. For each of the faulty versions, 5 repetitions were made, to a total of 500 prioritized test suites per program.

B. Input Parameters

For our study, coverage matrices A were obtained by instrumenting each of the programs at statement level with `Zoltar`. It must be taken into account that the coverage of a test input can vary between regression cycles. This will not affect diagnostic accuracy as diagnosis is performed *a posteriori* when the updated coverage is already available. However, it can affect the accuracy of prioritization heuristics such as FEP, ART, IG or RAPTOR because the coverage of a test case is needed *a priori*. If using the coverage information from a previous test run, the coverage can deviate (usually slightly). This deviation should be taken into account, by using techniques for estimating the updated coverage of a test input [9]. This situation is generally overlooked in test prioritization literature [15], [24], [35].

Regarding the input parameters of Bayesian diagnosis. It is safe to assume equal-valued priors $p_j = 0.01$ (10 defects/K-LOC), since priors have been shown not to be very critical to diagnostic performance [4]. On the other hand, the accuracy of the diagnosis itself depends on accurate FNR estimations [4], [10]. In our experiments, we will derive FNR information from *testability*, by using a simplified propagation, infection, execution (PIE) technique [40] used also in [16], [35].

C. Fault Localization Effectiveness

We compare diagnostic prioritization effectiveness in terms of diagnostic effectiveness S, according to Equation 1.

1) Statistical Diagnosis (Ochiai SC): Table IV presents the numerical results in terms of the S score and relative percentage with respect to RND. The best techniques with 95% statistical confidence in the Bonferroni mean separation test are highlighted in boldface.

The SEQUOIA column is obtained by using the test order produced by SEQUOIA, discarding its internal Bayesian diagnosis and performing a new diagnosis using the similarity coefficient. It can be seen how SEQUOIA's effectiveness appears to be extremely poor, in contrast with its performance when using Bayesian diagnosis as shown in the following section. This is caused by the fact that similarity coefficients do not exploit all the information available.

It can also be seen how RAPTOR consistently provides an improvement in the efficiency for all programs with only one fault, and is the best in all cases but one (`tcas`). On average, RAPTOR provides 54% reduction in terms of S with respect to RND's baseline performance in the single-fault case, and 45% in the multiple-fault case. When compared to the next best technique, ART, RAPTOR provides 38% reduction in the single fault case on average, and 15% reduction in the multiple fault case. For multiple faults, RAPTOR's superiority is still clear, although there are some cases where the performance of RAPTOR is severely affected because the distribution of the faults is very biased, as will be explained in Section VI.

For the Siemens programs, ADDST and FEP perform extremely poorly even when compared to RND. However, for the SIR programs the trend reverses, producing much better results. The root causes for the low effectiveness of RAPTOR for `tcas`, and the striking difference between the effectiveness of ADDST for Siemens and SIR programs will be analyzed in Section VI.

2) Bayesian Diagnosis: We now evaluate the effectiveness of RAPTOR and other techniques in terms of Bayesian fault diagnosis. Table V presents the numerical results, in terms of the S score. Again we present the reduction percentage with

	Ochiai, $M_f = 1$										
	RND	ART		ADDST		FEP		RAPTOR		SEQUOIA	
pt	3.86	2.46	-36%	10.06	161%	12.09	214%	**1.56**	**-60%**	7.05	83%
p2	4.90	3.21	-35%	8.43	72%	14.69	200%	**2.25**	**-54%**	6.48	32%
re	4.64	5.19	12%	10.99	137%	12.01	159%	**3.44**	**-26%**	10.31	122%
sc	6.19	6.34	2%	17.85	188%	17.71	186%	**3.77**	**-39%**	12.06	95%
s2	1.77	2.11	19%	8.15	361%	9.03	411%	**1.30**	**-26%**	6.63	275%
tc	3.81	**3.21**	**-16%**	11.53	203%	13.02	242%	3.47	-9%	5.20	37%
ti	2.20	1.91	-13%	4.96	126%	7.93	261%	**1.68**	**-24%**	7.57	245%
sp	13.20	10.61	-20%	8.21	-38%	9.20	-30%	**7.55**	**-43%**	14.95	13%
gz	12.98	7.11	-45%	2.81	-78%	2.88	-78%	**2.43**	**-81%**	12.60	-3%
se	10.83	5.35	-51%	4.15	-62%	4.32	-60%	**1.83**	**-83%**	12.03	11%
gr	12.05	8.44	-30%	8.18	-32%	9.71	-19%	**6.43**	**-47%**	15.02	25%
fl	4.47	1.71	-62%	1.40	-69%	1.55	-65%	**0.94**	**-79%**	3.71	-17%

	Ochiai, $M_f = 3$										
	RND	ART		ADDST		FEP		RAPTOR		SEQUOIA	
pt	4.52	**2.01**	**-55%**	9.70	115%	6.64	47%	3.33	-26%	7.74	71%
p2	4.96	2.45	-51%	4.77	-4%	6.07	22%	**1.42**	**-71%**	5.42	9%
re	4.38	4.77	9%	5.65	29%	7.07	61%	**2.49**	**-43%**	12.94	195%
sc	7.55	5.20	-31%	**1.29**	**-83%**	9.22	22%	7.37	-2%	9.94	32%
s2	2.78	**1.27**	**-54%**	7.31	163%	4.51	62%	2.88	3%	**1.29**	**-54%**
tc	2.37	**2.27**	**-4%**	13.10	453%	13.79	482%	3.68	55%	5.86	147%
ti	6.73	4.59	-32%	3.13	-54%	4.50	-33%	**1.15**	**-83%**	6.81	1%
sp	12.91	8.76	-32%	6.60	-49%	7.97	-38%	**5.76**	**-55%**	18.77	45%
gz	5.28	3.52	-33%	**2.14**	**-59%**	3.71	-30%	**2.35**	**-56%**	4.22	-20%
se	6.79	3.26	-52%	2.81	-59%	4.78	-30%	**1.77**	**-74%**	8.96	32%
gr	10.95	4.72	-57%	7.39	-33%	7.96	-27%	**4.26**	**-61%**	14.26	30%
fl	3.62	2.33	-36%	3.60	-1%	4.26	+18%	2.32	-36%	**1.42**	**-61%**

TABLE IV
EFFECTIVENESS (S) IN TERMS OF STATISTICAL DIAGNOSIS (OCHIAI)

	Bayes, $M_f = 1$										
	RND	ART		ADDST		FEP		RAPTOR		SEQUOIA	
pt	3.79	2.62	-31%	10.12	167%	13.00	243%	2.20	-42%	**1.60**	**-58%**
p2	3.23	1.83	-43%	6.46	100%	14.23	340%	**1.03**	**-68%**	1.51	-53%
re	2.70	1.88	-30%	3.87	43%	3.99	48%	**1.61**	**-40%**	2.88	7%
sc	1.57	2.00	27%	7.39	369%	11.79	650%	**1.30**	**-17%**	1.69	7%
s2	2.41	2.02	-16%	7.91	229%	10.95	355%	2.37	-2%	**1.02**	**-58%**
tc	3.00	**2.39**	**-20%**	8.82	194%	10.49	249%	3.60	20%	10.02	234%
ti	**1.28**	**1.32**	**3%**	6.38	401%	8.52	568%	4.97	290%	3.88	205%
sp	7.13	5.77	-19%	**2.51**	**-65%**	3.25	-54%	3.24	-55%	3.06	-57%
gz	8.08	4.06	-50%	**1.54**	**-81%**	1.75	-78%	**1.55**	**-81%**	2.24	-72%
se	5.72	3.00	-48%	2.04	-64%	2.23	-61%	**1.38**	**-76%**	1.93	-66%
gr	6.21	3.86	-38%	3.04	-51%	3.57	-43%	**2.70**	**-57%**	3.91	-37%
fl	3.40	1.69	-50%	**1.09**	**-68%**	**1.12**	**-67%**	1.18	-65%	1.41	-59%

	Bayes, $M_f = 3$										
	RND	ART		ADDST		FEP		RAPTOR		SEQUOIA	
pt	5.71	6.37	11%	25.69	350%	27.38	379%	**3.31**	**-42%**	5.49	-4%
p2	8.56	6.51	-24%	21.09	146%	27.77	224%	**3.81**	**-55%**	5.74	-33%
re	10.95	8.84	-19%	10.36	-5%	11.79	8%	**5.08**	**-54%**	**5.12**	**-53%**
sc	13.07	9.25	-29%	9.72	-26%	21.83	67%	**4.95**	**-62%**	5.81	-56%
s2	11.78	9.07	-23%	9.79	-17%	19.81	68%	3.06	-74%	**2.70**	**-77%**
tc	4.44	**3.94**	**-11%**	13.16	196%	14.88	235%	7.66	72%	12.53	182%
ti	3.00	2.20	-27%	3.65	22%	10.06	235%	**1.31**	**-56%**	4.43	48%
sp	12.48	10.04	-20%	7.48	-40%	**6.64**	**-47%**	7.03	-44%	8.33	-33%
gz	14.07	7.14	-49%	**2.03**	**-86%**	**1.95**	**-86%**	**2.64**	**-81%**	3.69	-74%
se	10.88	6.49	-40%	**3.40**	**-69%**	4.62	-58%	**3.39**	**-69%**	3.82	-65%
gr	10.78	8.22	-24%	**3.68**	**-66%**	3.77	-65%	4.87	-55%	6.52	-40%
fl	9.90	5.04	-49%	3.63	-63%	4.09	-59%	3.41	-66%	**3.26**	**-67%**

TABLE V
EFFECTIVENESS (S) IN TERMS OF BAYESIAN DIAGNOSIS

respect to RND, and the scores in boldface correspond to the top techniques with 95% confidence following the Bonferroni mean separation test.

Combined with Bayesian diagnosis, SEQUOIA is no longer among the worst performing algorithms but among the best, showing the perfect match between the heuristic and the diagnosis algorithm. However, it can be seen how RAPTOR's effectiveness is on par and even surpasses SEQUOIA, the theoretical optimum, both in the single-fault and multiple-fault case. This is due to the fact that the IG heuristic used in SEQUOIA is largely affected by errors in the FNR estimations, whereas RAPTOR is not.

On average, RAPTOR provides a 40% reduction of S in the single fault case with respect to the baseline set by RND. In the multiple fault case, the reduction with respect to RND is 55%. When compared to the next best technique, ART, RAPTOR achieves a reduction of 13% for single faults, and 39% for multiple faults.

Identical to what happened in the similarity coefficient case, there is a noticeable difference in the effectiveness of FEP and ADDST, between the Siemens and SIR programs, which will be explained in Section VI.

D. Fault Detection (APFD$_c$)

We evaluate the techniques in terms of their fault detection performance, by using the APFD$_c$ metric [13]. This information is useful to understand the modest trade-off between the test prioritization goals of fault detection and fault localization. Consistently with previous literature [24], [35], we will use only single-fault data.

Table VI shows the results in terms of the improvement percentage of the APFD$_c$ metric with respect to RND. The values in boldface represent the best technique, with 95% confidence using the Bonferroni mean separation test.

Consistent with previous literature [24], [35], it can be seen how ADDST and FEP are the best performing techniques, especially in the Siemens programs. For the larger SIR programs, however, RAPTOR has the best performance, tied with ADDST.

Both observations are consistent with our previous findings [19] indicating that prioritizing for fault localization (independently of the specific heuristic) tends to lower failure rates in some test suites. This reduced performance is not a very serious problem in practice, given the 2% difference between ADDST and RAPTOR. Only `schedule2` presents a significant reduction of 9% for RAPTOR with respect to ADDST. However, the increased test effort is completely outweighed by the large savings in inspection cost.

	RND	ART		ADDST		FEP		RAPTOR		SEQUOIA	
pt	0.93	0.96	+3%	**0.98**	+5%	0.96	+2%	**0.97**	+4%	**0.97**	+4%
p2	0.94	0.96	+2%	**0.98**	+5%	**0.99**	+6%	0.96	+3%	**0.98**	+5%
re	0.89	0.89	+0%	0.88	-1%	**0.92**	+3%	**0.92**	+2%	**0.93**	+4%
sc	**0.95**	**0.95**	-0%	0.91	-4%	**0.95**	-0%	0.93	-2%	**0.94**	-1%
s2	**0.92**	**0.93**	+1%	**0.93**	+1%	0.90	-2%	0.87	-5%	**0.92**	+0%
tc	0.81	**0.84**	+3%	0.80	-2%	0.79	-3%	**0.85**	+5%	0.80	-1%
ti	0.97	0.97	-0%	**0.99**	+2%	0.98	+0%	0.96	-2%	**0.99**	+1%
sp	0.70	0.76	+9%	0.79	+13%	0.77	+11%	**0.82**	+17%	**0.79**	+13%
gz	0.74	0.86	+16%	**0.95**	+29%	**0.95**	+29%	**0.96**	+29%	**0.95**	+29%
se	0.75	0.90	+19%	0.94	+24%	0.94	+25%	**0.97**	+30%	0.94	+24%
gr	0.75	0.81	+8%	**0.83**	+11%	0.81	+8%	**0.85**	+14%	0.82	+9%
fl	0.94	0.98	+4%	0.98	+4%	0.98	+4%	**0.99**	+5%	**0.99**	+5%

TABLE VI
FAULT DETECTION PERFORMANCE (APFD$_c$)

VI. Discussion

In this section we discuss the threats to the validity of our results, analyze the reasons for the behavior of the techniques, and introduce some criteria aimed at choosing the right technique under varying circumstances.

A. Threats to Validity

The external validity of the results obtained with the Siemens programs can be questioned given their small sizes. In order to strengthen the validity of our results we provided validation experiments with larger programs such as the SIR tools, showing that indeed, for larger matrices the behavior of techniques is different.

The fact that we seeded artificial faults into our programs is a second threat to the external validity of our results. From the point of view of the representativeness of mutation faults, they have become usual in test prioritization and fault diagnosis literature [35], [41], since it is almost impossible to obtain a sample of real faults that is representative. From the fault density point of view, we restricted the experiment to 3 simultaneous faults to keep fault density to a realistic level.

A number of other threats to the validity of our experiments have been discussed throughout the paper. The motivation and construct validity of the S metric was discussed in Section II. Using two different diagnosis techniques to measure efficiency was motivated in Section V. The threats to validity derived from how we determine the algorithm input parameters, especially the usage of up-to-date coverage information, the prior fault probabilities and false negative rates, all were covered in Section V.

B. Technique Analysis

It can be seen in our results that there are significant differences in the effectiveness of each technique depending on the program. There is also a noticeable difference between the Siemens programs and the SIR programs that influences the behavior of ADDST, transforming it from the worst of all techniques, to a very competitive one.

To explain the difference in the effectiveness of ADDST and FEP between Siemens and SIR programs, it is necessary to explain the relationship between coverage and information gain. For simplicity of exposition, let us assume that there is only one fault in the system, and that the fault will produce a failure whenever covered (FNR=0%). Let us also assume that the coverage of a test case is uniformly distributed throughout the program. In this situation, the fault will always be at the top of the suspect ranking, possibly tied to other candidates. In this simplified case, the information gain that a test provides is determined by the reduction of the size of the top-ranked suspect set. Assuming there are $|D|$ top-ranked suspects, a test of coverage density ρ will reduce the top-ranked set to $|D| \cdot \rho$ statements if it fails, and $|D| \cdot (1-\rho)$ if it passes.

Under these conditions, it is shown in [18], that diagnostic information gain can be modeled by

$$IG(\rho) = -\rho \log_2 \rho - (1-\rho) \log_2 (1-\rho) \tag{7}$$

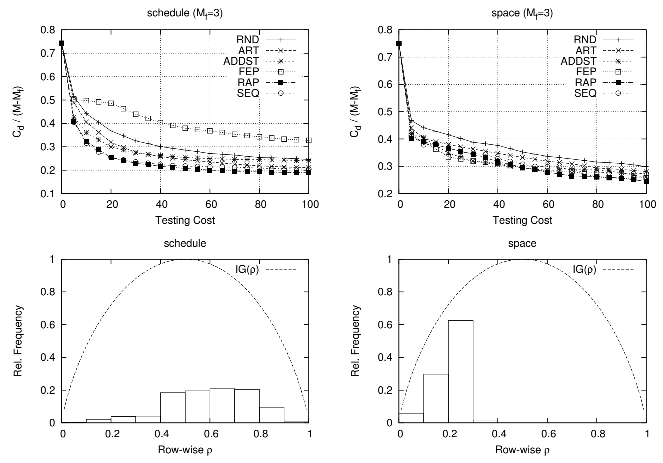

Fig. 2. Impact of coverage density distribution

which is optimal for $\rho = 0.5$ (50% coverage probability, which also corresponds to 50% failure probability in this case).

In real test matrices coverage density ρ is not uniform, but varies for each test. The plot in Figure 2 depicts the evolution of C_d and coverage density histogram of two programs from our evaluation. The IG curve is added as reference. Techniques like ADDST and FEP, designed to *maximize* coverage, will tend to select test cases with a very high ρ (thus, a high failure probability) if they are available, which will fall *past* the IG maximum (0.5). This is the case for the `schedule` program. This affects their effectiveness greatly, as can be seen in our results. However, when there are no tests past the IG maximum (very sparse matrices), as is the case for `space` and the other SIR programs, ADDST and FEP are indirectly choosing tests that are close to the optimum by maximizing coverage.

This coverage distribution also explains the difference in the APFD$_c$ score between Siemens and the SIR programs. For the Siemens set, Raptor and Sequoia *lower* the coverage, whereas for the SIR programs, they both maximize it, as ADDST does, hence the similar scores.

The bad results of Raptor for the `tcas` program are due to the program's nature and some subtle assumptions underlying Raptor. First, `tcas` has 174 lines of code but only 9 ambiguity groups, which severely impacts the performance of any similarity coefficient and makes any improvement in effectiveness very small. Second, our sample of seeded faults in `tcas` and `tot_info` are not uniformly distributed in the code, making Raptor less optimal. Sequoia, being a dynamic technique, can adapt to this non-uniformity, but the errors in the FNR estimation (which affect the IG heuristic) neutralize the gain.

C. Choice of Technique

We now analyze the factors that influence the decision to test for failures or faults from the first test, and to use either Raptor, Sequoia, or an alternative.

Wait for the first failure? We have seen in our evaluation that the fault detection cost, i.e., the test effort it takes to detect

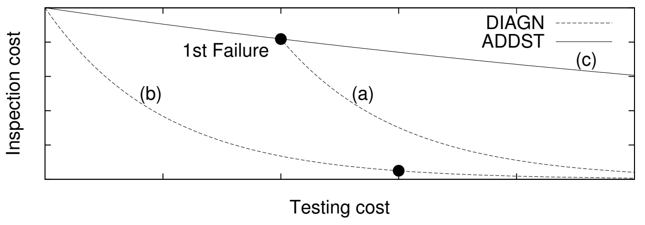

Fig. 3. Prioritization techniques and usage scenarios

the first failure, can be slightly increased when using diagnostic test prioritization. The first question is whether to perform diagnostic test prioritization from test 0, or only after the first failure, since RAPTOR and SEQUOIA have a moderately decreased fault detection capability. The most conservative solution, especially if the introduction of a fault is unlikely, is to initially use ADDST or FEP, and switch to diagnostic test prioritization if a failure occurs. This corresponds to scenario (a) in Figure 3. However, if the probability of introducing a fault during development is high (some projects can have up to 70% probability [27]), it is better to start directly with diagnostic test prioritization since the tests executed until the first failure is found provide the most valuable diagnostic information. The increased test cost (the first failure occurs slightly later) is greatly outweighed by the much more substantial gains in terms of diagnostic performance per unit of test effort. Therefore, even if it takes slightly longer to *detect* a fault, the reduction in the inspection effort required to precisely *locate* the fault greatly compensates for this. This corresponds to scenario (b). Furthermore if coverage density ρ is very low, diagnostic prioritization and FEP or ADDST will provide similar fault detection and localization capabilities to SEQUOIA or RAPTOR. This corresponds to scenario (c).

RAPTOR or SEQUOIA? If using low-cost similarity coefficients, RAPTOR is the clear choice. However, since SEQUOIA can potentially provide better results than RAPTOR combined with Bayesian diagnosis, it is necessary to consider the advantages and disadvantages of each technique.

1) **Computational cost:** Even though RAPTOR and SEQUOIA have a similar order of complexity, RAPTOR is based on fast bit-wise operators, whereas SEQUOIA is based on expensive floating point operations, and requires expensive prior parameter determination. Furthermore, it must be taken into account that SEQUOIA must perform on-line prioritization, as the tests are executed, potentially introducing a significant overhead.
2) **FNR estimation quality:** the IG heuristic used in SEQUOIA can be severely affected by poor FNR estimations. If the variance of the samples used to estimate each h_j is high (as is the case for `tcas`), one should not consider SEQUOIA.
3) **Ambiguity and non-uniformity:** a test matrix with very few and very large ambiguity groups, or a system with extremely biased fault distribution can cause problems to a static algorithm like RAPTOR. If the FNR estimation quality is good enough, one could opt for SEQUOIA. Otherwise, the best would be to opt for a random approach such as ART or RND.

VII. RELATED WORK

The influence of test-suite extension, reduction, modification, and prioritization on fault detection and diagnosis has received considerable attention [20], [25], [44]. All the works cited study the effect of prioritization and reduction techniques that were designed for fault detection, on fault localization. RAPTOR, on the other hand, is designed with fault localization effectiveness in mind, to optimize the improvement of diagnostic accuracy per unit of test effort.

Baudry *et al.* [7] propose Test for Diagnosis (TfD) to evaluate the quality of a test suite for diagnosis. Their measurement is based also on ambiguity groups (*dynamic basic blocks* (DBB), in their work). However, their heuristic is focused exclusively on the number of ambiguity groups, not on their size. This may lead to needlessly sparse matrices (e.g., a diagonal matrix in their terms has ideal TfD fitness, but at excessive test cost). Our algorithm recursively dissects ambiguity groups using a much more sophisticated goal function. As a result, tests are selected that offer higher diagnostic performance per test, optimizing the coverage overlap in tests which allows deducing the defect locations in less tests. Finally, our algorithm handles multiple defects while [7] only considers single faults, a highly unrealistic assumption in large systems.

Test case prioritization's most common goal is to increase failure detection rate [21], [43]. Multiple coverage-based prioritization techniques have been proposed and studied [14], [24], [28], [35], including techniques that take testing cost variance into account [13], [38], [46]. RAPTOR fundamentally differs from the above prioritization techniques in that its main goal is not to optimize the rate of failure detection, but to also minimize debugging effort.

Automated fault-localization techniques aim at minimizing residual diagnostic effort when failures occur during the testing phase. Statistical approaches include [1], [26], [29], [30], [34], [42]. A recent, probabilistic approach of acceptable complexity is [4]. Other approaches to fault localization include [6], [36], [41]. All the above approaches do not address test sequencing to optimize diagnostic accuracy, and are essentially similar to RND.

Sequential diagnosis aims at finding the test sequence that optimizes diagnostic performance based on the current test outcomes, applied to hardware systems [33]. If the system can have multiple faults, sequential diagnosis complexity becomes exponential, and can only be approximated [17], [37]. The intelligent probing mechanism by Brodie *et al.* [8] uses a similar measurement of ambiguity as RAPTOR, for test matrix reduction (minimal set of probes), applied to computer networks.

VIII. CONCLUSIONS & FUTURE WORK

In this paper we presented a diagnostic test prioritization algorithm, RAPTOR, that selects test cases by their ability to produce a refined diagnosis (fault *localization*) rather than to produce a failure as early as possible (fault *detection*). Our results show that RAPTOR's test prioritization can significantly improve the reduction of inspection cost per unit of test cost, when compared to random (RND), adaptive-random (ART), statement coverage (ADDST, FEP), and IG-based (SEQUOIA) prioritization. When using similarity coefficients, RAPTOR provides a 38% and 15% average reduction in S with respect to the next-best (ART) for single and multiple faults, respectively. In the case of Bayesian diagnosis, RAPTOR achieves a 13% and 39% average reduction of S with respect to the next-best (ART) for single and multiple faults, respectively. Given its negligible sacrifice in fault detection capability, RAPTOR can be used as the sole test prioritization strategy during regression testing in many cases, especially when the probability of introducing defects is high.

Future work focuses on modeling the influence of program and test suite characteristics on diagnosability. An understanding of these factors will allow us to define new criteria for the creation of new diagnostic test suites or to improve existing ones.

ACKNOWLEDGEMENTS

This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK03021 program and the POSEIDON project of the Embedded Systems Institute (ESI), Eindhoven, The Netherlands.

REFERENCES

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] R. Abreu, P. Zoeteweij, and A. van Gemund. An observation-based model for fault localization. In *Proc. WODA'08*.

[3] R. Abreu, P. Zoeteweij, and A. van Gemund. On the accuracy of spectrum-based fault localization. In *Proc. TAIC PART'07*.

[4] R. Abreu, P. Zoeteweij, and A. van Gemund. Spectrum-based multiple fault localization. In *Proc. ASE'09*.

[5] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proc. ICSE'05*.

[6] G. K. Baah, A. Podgurski, and M. J. Harrold. The probabilistic program dependence graph and its application to fault diagnosis. In *Proc. ISSTA'08*.

[7] B. Baudry, F. Fleurey, and Y. L. Traon. Improving test suites for efficient fault localization. In *Proc. ICSE'06*.

[8] M. Brodie, I. Rish, and S. Ma. Intelligent probing: a cost-effective approach to fault diagnosis in computer networks. *IBM Syst. J.*, 41:372–385, 2002.

[9] P. K. Chittimalli and M. J. Harrold. Recomputing coverage information to assist regression testing. *IEEE Trans. Softw. Eng.*, 35(4):452–469, 2009.

[10] J. de Kleer. Diagnosing multiple persistent and intermittent faults. In *Proc. IJCAI'09*.

[11] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artif. Intell.*, 1987.

[12] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Emp. Soft. Eng. J.*, 10(4), 2005.

[13] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proc. ICSE'01*.

[14] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Soft. Quality Control*, 2004.

[15] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE TSE*, 28(2), 2002.

[16] A. Gonzalez-Sanchez, R. Abreu, H.-G. Gross, and A. J. van Gemund. An empirical study on the usage of testability information to fault localization in software. In *Proc. SAC'11*.

[17] A. Gonzalez-Sanchez, R. Abreu, H.-G. Gross, and A. J. van Gemund. Spectrum-based sequential diagnosis. In *Proc. AAAI'11*. (To appear).

[18] A. Gonzalez-Sanchez, H.-G. Gross, and A. J. van Gemund. Modeling the diagnostic efficiency of regression test suites. In *Proc. TEBUG'11 (ICST'11 Workshop)*.

[19] A. Gonzalez-Sanchez, E. Piel, R. Abreu, H.-G. Gross, and A. J. van Gemund. Prioritizing tests for software fault localization. *Software: Practice and Experience*, 2011. Accepted for publication.

[20] D. Hao, L. Zhang, H. Zhong, H. Mei, and J. Sun. Eliminating harmful redundancy for testing-based fault localization using test suite reduction: An experimental study. In *Proc. ICSM'05*.

[21] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM TSEM*, 2(3), 1993.

[22] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. ICSE '94*.

[23] T. Janssen, R. Abreu, and A. J. C. van Gemund. ZOLTAR: A toolset for automatic fault localization. In *Proc. ASE'09 - Tool Demonstrations*.

[24] B. Jiang, Z. Zhang, W. Chan, and T. Tse. Adaptive random test case prioritization. In *Proc. ASE'09*.

[25] B. Jiang, Z. Zhang, T. H. Tse, and T. Y. Chen. How well do test case prioritization techniques support statistical fault localization. In *Proc. COMPSAC'09*.

[26] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proc. ICSE'02*.

[27] S. Kim, E. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE TSE*, 34(2):181 –196, march-april 2008.

[28] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE TSE*, 33(4), 2007.

[29] B. Liblit. Cooperative debugging with five hundred million test cases. In *Proc. ISSTA'08*.

[30] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. Sober: Statistical model-based bug localization. In *Proc. ESEC/FSE-13*.

[31] W. Mayer and M. Stumptner. Evaluating models for model-based debugging. In *ASE'08*.

[32] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM TOSEM*, 5:99–118, April 1996.

[33] V. Raghavan, M. Shakeri, and K. R. Pattipati. Test sequencing algorithms with unreliable tests. *IEEE TSMC*, 29(4), 1999.

[34] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Proc. ASE'03*.

[35] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE TSE*, 27(10), 2001.

[36] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold. Lightweight fault-localization using multiple coverage types. In *Proc. ICSE'09*.

[37] M. Shakeri, V. Raghavan, K. R. Pattipati, and A. Patterson-Hine. Sequential testing algorithms for multiple fault diagnosis. *IEEE TSMC*, 2000.

[38] A. M. Smith and G. M. Kapfhammer. An empirical study of incorporating cost into test suite reduction and prioritization. In *Proc. SAC'09*.

[39] G. Stenbakken, T. Souders, and G. Stewart. Ambiguity groups and testability. *IEEE Trans. on Instr. and Meas.*, 38(5):941 –947, Oct. 1989.

[40] J. M. Voas. Pie: A dynamic failure-based technique. *IEEE TSE*, 18(8):717–727, 1992.

[41] X. Wang, S. Cheung, W. Chan, and Z. Zhang. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In *Proc. ICSE'09*.

[42] W. Wong, T. Wei, Y. Qi, and L. Zhao. A crosstab-based statistical method for effective fault localization. In *Proc. ICST'08*.

[43] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. ISSRE'97*.

[44] Y. Yu, J. A. Jones, and M. J. Harrold. An empirical study of the effects of test-suite reduction on fault localization. In *Proc. ICSE'08*.

[45] A. Zeller. Isolating cause-effect chains from computer programs. In *Proceedings of Symposium on the Foundations of Software Engineering (FSE'02)*.

[46] L. Zhang, S.-S. Hou, C. Guo, T. Xie, and H. Mei. Time-aware test-case prioritization using integer linear programming. In *Proc. ISSTA '09*.