Delft University of Technology
Software Engineering Research Group
Technical Report Series

Improving the Software Fault Localization Process through Testability Information

Alberto Gonzalez-Sanchez, Rui Abreu, Hans-Gerhard Gross
and Arjan van Gemund

Report TUD-SERG-2010-011

TUD-SERG-2010-011

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
`http://www.se.ewi.tudelft.nl/techreports/`

For more information about the Software Engineering Research Group:
`http://www.se.ewi.tudelft.nl/`

Note: Submitted for review at ICTSS'10

© copyright 2010, by the authors of this report. Software Engineering Research Group, Department of Software Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology. All rights reserved. No part of this series may be reproduced in any form or by any means without prior written permission of the authors.

Improving the Software Fault Localization Process through Testability Information*

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

[1] Software Technology Department, Delft University of Technology, The Netherlands
{a.gonzalezsanchez,h.g.gross,a.j.c.vangemund}@tudelft.nl
[2] Department of Informatics Engineering, University of Porto, Portugal
rui@computer.org

Abstract. When failures occur during software testing, automated software fault localization helps to diagnose their root causes and identify the defective components of a program to support debugging. Diagnosis is carried out by selecting test cases in such way that their pass or fail information will narrow down the set of fault candidates, and, eventually, pinpoint the root cause. An essential ingredient of effective and efficient fault localization is the knowledge about the intermittency of occurring failures, i.e., the rate at which defective components of a program will exhibit failures. In current fault localization processes, intermittency is either ignored completely, or merely estimated *a posteriori* as part of the diagnosis. In this paper, we study the reduction in testing and diagnosis effort when intermittency is known *a priori*. We deduce intermittency from testability, following the propagation-infection-execution (PIE) approach. Experiments with synthetic and real programs suggest significant improvement in the combined testing and diagnosis process. When compared to the next best technique, testability-based intermittency information reduces the average number of tests required to reach the same diagnosis quality by 55%, and provides an effort reduction for fault localization of 30% for the same testing effort.

1 Introduction

Testing is the most commonly used method for detecting the presence of faults in software. However, once the presence of a fault has been detected (by means of a failing test), its precise location has to be determined. Fault localization denotes the process of finding the root-cause of failures through diagnosis to support debugging. Diagnostic accuracy is a critical success factor in the cycle of testing, diagnosing, acting/recovering, and repairing. However, typically multiple diagnoses are possible, and further tests are executed to narrow down the set of possibilities, ideally until a perfectly accurate diagnosis is reached.

* This work has been carried out as part of the Poseidon project under the responsibility of the Embedded Systems Institute (ESI). This project is partially supported by the Dutch Ministry of Economic Affairs under the BSIK03021 program.

In many practical situations, faults manifest themselves intermittently, i.e., not all tests of a component will produce a failure. Ignoring intermittency as done in statistical diagnosis approaches [1, 2], results in degraded diagnostic performance [3]. An alternative approach is to approximate the intermittency rates during the diagnosis process [4, 5]. However, the absence of prior knowledge of component intermittency rates leads to a loss of diagnostic accuracy. The diagnosis algorithm must approximate the intermittency rates while it computes the diagnosis, and that, again, depends on the intermittency rates. The net result is that a large number of tests is required to obtain an accurate estimate of intermittency, thereby reducing the rate at which the diagnosis converges, and thus, increasing both testing effort and diagnostic effort.

In software fault localization, intermittency information can be derived from testability, and determined *a priori*, using testability quantification techniques. An example is the work on estimating the propagation-infection-execution (PIE) probability by Voas and Miller [6], or the simpler so-called failure exposing potential by Rothermel [7]. By providing testability-based intermittency information, the intermittency estimation problem can be detached from the diagnosis, leading to significant performance gains in the fault localization process.

In this paper we evaluate to which extent testability information can lead to performance gains in the fault localization process. We study and assess the testability of the components in a system, and use that information as input to a Bayesian diagnosis algorithm. In particular, the paper makes the following contributions.

1. We compare the performance of Bayesian diagnosis with prior intermittency information with diagnosis methods that do not exploit this prior information.
2. We perform the comparison for synthetic test data where component intermittency can be controlled, as well as other parameters such as component involvement frequencies, problem size, etc.
3. We also perform the comparison for empirical test data obtained from the Siemens test suite, and the SIR program `space`. Statement and function intermittency data is obtained by using the intermittency data produced by mutation analysis [6, 7].

Our results show that the gains achieved by introducing prior knowledge about fault intermittency are significant. For real programs containing real faults, we observed 55% average reduction on the number of test cases to reach the same diagnosis quality, and 30% diagnostic performance improvement for the same testing effort.

The paper is organized as follows. Section 2 presents the the problems posed by intermittency, and its influence in the software domain. Section 3 introduces the main concepts of diagnosis and the main diagnostic techniques. A theoretical and empirical validation is performed in Sections 4 and 5 respectively. Section 6 describes related work in the area of fault diagnosis. Section 7 concludes the paper and presents our future research directions.

2 The Need for Intermittency Information

In probabilistic diagnosis, information on the system (nominal or faulty behavior) is encoded in terms of probabilistic component models, and exploited to reason about the health states of the components given a sequence of system observations. Recent research [3] has shown that in order for a probabilistic approach to work in practice, these models need to be able to express *intermittency*, i.e., the fact that for the same input (working condition) a faulty component's output can alternate between correct (nominal behavior) and incorrect (faulty behavior). Obvious examples in hardware are a communication channel that occasionally flips a data bit, or a copier where sometimes sheets may be blank, or where a worn roller sometimes slips and causes a paper jam [8].

However, intermittency information is hardly ever known *a priori*, which leads to a significant loss of diagnostic performance. For example, in sequential fault diagnosis [9] and test prioritization [7] the tests are ordered such that a minimal number of test cases will lead to an adequate diagnosis. This requires a heuristic function to estimate the failure probability and gain in diagnosis information per test case executed. Without prior intermittency information, current heuristics cannot deliver sufficient precision to achieve competitive convergence rates towards a diagnosis, when compared to arbitrary test ordering [10, 11]. Quick convergence towards a diagnosis is utterly essential when tests are extremely expensive to perform. Despite the strong need for such diagnosis techniques, no technique has emerged that exploits prior knowledge on intermittency data, delivering competitive convergence.

Fault Intermittency in Software. Software is in most cases inherently deterministic at the input level. Excluding situations like race conditions or truly non-deterministic behavior, the same input will always produce the same output. However, during diagnosis, information about the inputs is abstracted away, and tests are modeled on higher levels of abstraction related to software components, e.g., statements, functions, modules, classes. If a component is tested with multiple inputs, from the diagnosis point of view, only the facts "the component is tested" and "the component passes or fails" are considered. Not all the inputs will produce a failure, because of the different paths taken in a test, so that this abstraction introduces an apparent intermittent behavior for software diagnosis.

As a simple example, consider the integer division statement `y = x / 10` where `10` is a fault that should have been `15`. Consider two input values `x = 15`, and `x = 20`, which should both produce `y = 1` as output. In the first case, the component produces a correct output ($15/10 = 1$), whereas in the second case the component fails ($20/10 = 2$). In an abstract modeling approach at statement level, both inputs are abstracted to "the component is tested". The division component exhibits intermittent failure behavior, i.e., when it is tested, the output alternates between "correct" and "not correct".

In the previous example, the number of inputs where the output is "correct" is restricted to a subset of the $[0, 30]$ integer interval. If inputs can be drawn randomly from the 2^8 possibilities for a byte, there are approximately 90% "not correct" outputs, or, abstracted to the statement level, the statement fails with an intermittency of 0.1 (i.e., one out of 10 tests does not fail). Next, consider the boolean statement `y = x < 10`, which is faulty and should be `y = x <= 10`. There is only one input, `10`, that will produce a failure. This fault's intermittency is therefore $1 - 1/256 = 0.996$.

Testability. Fault intermittency in software diagnosis is intimately connected with the concept of *testability*. Of the multiple definitions of testability, the one proposed by Voas and Miller [12] is the closest to the concept of intermittency: *the degree to which software reveals faults during testing*. This relates to the number of tests to be executed on a faulty component before it exposes the fault as a failure. Domain Testability [13], Domain to Range Ratio (DRR) [14–16], and semantic fault size [17] have been proposed as static methods to testability quantification. These methods are related to the concepts underlying the two previous examples: the probability that an input will produce a correct output if a function is faulty, depending on the relative sizes of the domain and the range of the function. However, the current state of the art does not allow for a straightforward usage as probabilities. Furthermore, it does not take into account the fact that a faulty component producing an error does not necessarily propagate the error to the output of the component.

Testability can also be modeled by the so-called propagation, infection, execution approach (PIE) [6]. PIE measures the probability that a statement is executed, the probability that it will produce an erroneous state, and the probability that the erroneous state will propagate to the output. The PIE approach involves an expensive mutation analysis, however, its implementation is simple and can be automated. In this paper, we use a variation of the PIE approach to obtain an estimation of the testability of a component, which is presented in Section 5.

3 Fault Diagnosis

The objective of fault diagnosis is to pinpoint the precise location of the faults in a program (bugs) by observing the program's behavior given a number of tests. For the purpose of this paper, which focuses on the usage of testability information, we will assume at most one fault is present in the system under test in order to avoid unnecessary complexity.

The following inputs are usually involved in automated diagnosis:

- A finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_j, \ldots, c_M\}$ of M components (e.g., source code statements, function points, classes, etc.) which are potentially faulty.
- A finite set $\mathcal{T} = \{t_1, t_2, \ldots, t_i, \ldots, t_N\}$ of N tests with binary outcomes $O = (o_1, o_2, \ldots, o_i, \ldots, o_N)$, where $o_i = 1$ if test t_i failed, and $o_i = 0$ otherwise.
- A $N \times M$ coverage matrix, $A = [a_{ij}]$, where $a_{ij} = 1$ if test t_i involves component c_j, and 0 otherwise.

The output of fault localization, is a *diagnosis*, i.e., a list or a component *ranking* $R =< r_1, r_2, \ldots, r_M >$ of component indices, ordered by the *likelihoods* $L =< l_{r_1}, l_{r_2}, \ldots, l_{r_M} >$ of each component c_{r_m} being faulty, (i.e., $l_{r_m} \geq l_{r_{m+1}}$). In this paper we consider two approaches for obtaining the component likelihoods L, (1) a Bayesian approach, where the likelihoods are exact fault probabilities, and (2) a statistical approach, where the l_{r_m} are so-called similarity coefficients. The diagnosis ranking R is returned to the tester who typically verifies the faults by going through R in descending order. The diagnosis cost, C_d, is modeled by

$$C_d = \frac{|c_j : l_j > l_*| + |c_j : l_j \geq l_*| - 1}{2} \tag{1}$$

where l_* is the likelihood of the actual fault, c_*. This corresponds to the position of c_* in R, and represents the effort needlessly spent by the tester or the developer inspecting the c_j that were not defective (false positives), while going through R top-down until c_* is found [5]. Because multiple explanations can be assigned the same probability, the value of C_d is averaged between the explanations that share the same likelihood, amongst which the real fault c_* is located. The value of C_d can be normalized by dividing it by the number of healthy components, $C_d = C_d/(M-1)$, to be able to compare systems of different sizes in relative terms.

3.1 Statistical Fault Diagnosis

Statistical fault diagnosis takes the approach of ignoring intermittency, although the likelihood value obtained is still influenced by the intermittency of the fault, as shown in [5]. A well-known statistical approach to fault diagnosis that originates from the Software Engineering domain is Spectrum-based Fault Localization [1, 2]. Here, the likelihood l_j is quantified in terms of *similarity coefficients*. A similarity coefficient measures the statistical similarity between component 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 number of times a_{ij} and o_i form the combinations $(0, 0), \ldots, (1, 1)$, respectively, i.e.,

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

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. For example, the likelihood l_j can be calculated according to the Tarantula [2], and Ochiai [1] similarity coefficients, given by

$$\text{Tarantula:} \quad l_j = \frac{\dfrac{n_{11(j)}}{n_{11(j)} + n_{01(j)}}}{\dfrac{n_{11(j)}}{n_{11(j)} + n_{01(j)}} + \dfrac{n_{10(j)}}{n_{10(j)} + n_{00(j)}}} \tag{3}$$

$$\text{Ochiai:} \quad l_j = \frac{n_{11(j)}}{\sqrt{(n_{11(j)} + n_{01(j)}) \cdot (n_{11(j)} + n_{10(j)})}} \tag{4}$$

Despite their lower diagnostic accuracy [5], similarity coefficients have gained much interest. A great advantage of similarity coefficients is their ultra-low computational complexity compared to probabilistic approaches and their independence from prior knowledge of any kind. Therefore, we will include them in our evaluation.

3.2 Probabilistic Fault Diagnosis

As an alternative to similarity coefficients, a form of probabilistic fault localization can be used. Bayesian diagnosis is a probabilistic reasoning approach aimed at obtaining a set of diagnoses $D = \{d_1, \ldots, d_j\}$ explaining a condition. Each diagnosis d_j corresponds to a component c_j being the faulty one. Diagnoses are ranked according to their assumed accuracy, expressed in terms of a probability $\Pr(d_j)$. Initially, the probability of each diagnosis is $\Pr(d_j) = p_j$. This represents the *a priori* fault probability (“prior”) for a component c_j, i.e., the knowledge available before any test is executed, e.g., fault density. After each test case t_i, the probability of each diagnosis $d_j \in D$ is updated depending on the outcome o_i of the test, following Bayes’ rule:

$$l_j = \Pr(d_j|o_i, o_{i-1}, \ldots) = \frac{\Pr(o_i|d_j)}{\Pr(o_i)} \cdot \Pr(d_j|o_{i-1}, \ldots) \tag{5}$$

In this equation, $\Pr(d_j|o_{i-1}, \ldots)$ represents the prior probability of diagnosis d_j before the test is executed. $\Pr(o_i)$ is the probability of the observed outcome, independent of which diagnosis is the correct one. This normalizing factor is given by

$$\Pr(o_i) = \sum_{d_j \in D} \Pr(o_i|d_j) \cdot \Pr(d_j|o_{i-1}, \ldots) \tag{6}$$

$\Pr(o_i|d_j)$ represents the probability of the observed outcome o_i produced by a test t_i, if that diagnosis d_j was the correct one. For single faults, it is defined as

$$\Pr(o_i|d_j) = \begin{cases} 1 & \text{if } a_{ij} = 0 \text{ and } o_i = 0 \\ 0 & \text{if } a_{ij} = 0 \text{ and } o_i = 1 \\ h_j & \text{if } a_{ij} = 1 \text{ and } o_i = 0 \\ 1 - h_j & \text{if } a_{ij} = 1 \text{ and } o_i = 1 \end{cases} \tag{7}$$

The value h_j is the *intermittency* of the fault candidate, i.e., the probability that the component c_j will not produce a failure when tested, if it is faulty. h stands for *health* and has a value between 0 (permanently failing) and 1 (never failing). This value can be provided to the Bayesian algorithm, or estimated from already executed tests. If *a priori* intermittency data is not available, an important problem in using this model is the estimation of h_j. In [5] it was proven that the optimal strategy to estimate h_j assuming a single fault is:

$$h_j = \frac{n_{10}(j)}{n_{10}(j) + n_{11}(j)} \tag{8}$$

	Program: Character Counter	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	Prior
c_0		0	0	0	0	0	0	0	0	
c_1	`main() {`	1	1	1	1	1	1	1	1	0.08
c_2	`int let, dig, other, c;`	1	1	1	1	1	1	1	1	0.08
c_3	`let = dig = other = 0;`	1	1	1	1	1	1	1	1	0.08
c_4	`while(c = getchar()) {`	1	1	1	1	1	1	1	1	0.08
c_5	`if ('A'<=c && 'Z'>=c)`	1	1	1	1	1	1	1	0	0.08
c_6	`let += 2; /* FAULT */`	1	0	1	1	0	0	1	0	0.08
c_7	`elif ('a'<=c && 'z'>=c)`	1	1	0	1	1	1	1	0	0.08
c_8	`let += 1;`	1	0	0	0	1	0	1	0	0.08
c_9	`elif ('0'<=c && '9'>=c)`	1	1	0	1	1	1	0	0	0.08
c_{10}	`dig += 1;`	1	1	0	1	0	0	0	0	0.08
c_{11}	`elif (isprint(c))`	0	0	0	0	1	1	0	0	0.08
c_{12}	`other += 1;}`	0	0	0	0	1	0	0	0	0.08
c_{13}	`printf("%d %d %d\n", let, dig, others);}`	1	1	1	1	1	1	1	1	0.08
	Test case outcomes	1	0	1	1	0	0	1	0	

Table 1. Faulty program and Fault Diagnosis inputs.

where n_{10} and n_{11} are defined as in Equation 2.

A stable estimation requires a large number of tests, or it may be imprecise. This reduces the rate at which the diagnosis improves, requiring execution of a large number of tests to obtain a high quality diagnosis. In the remainder of this paper, we will show the gains of using testability estimations with different degrees of accuracy.

3.3 Example Diagnosis

We illustrate how similarity coefficients and Bayesian fault localization (with prior intermittency information) use the test information to produce a diagnosis. Table 1 shows an example faulty program [18], eight tests, and their statement coverage (the matrix A is transposed for the sake of readability).

Similarity Coefficients. After executing the first three tests, the counters for c_6 are $n_{11}(6) = 2$, $n_{10}(6) = 0$, $n_{01}(6) = 0$, $n_{00}(6) = 1$. Its likelihood being the faulty one according to *Ochiai* is $l_6 = 1.0$. The remaining components have lower likelihoods, as they all have $n_{10}(j) > 0$ or $n_{01}(j) > 0$. For example, for c_5, $l_5 = 0.82$. Components c_{11} and c_{12} are never covered, and therefore their likelihoods are 0.

Bayesian Diagnosis. After applying test t_1, we observe a failure. The probabilities of all the covered statements c_j (including c_6) are updated by $\frac{(1-h_{1,j})\cdot p_j}{\Pr(o_1)} = \frac{1\cdot 0.08}{0.85} = 0.09$. The statements which were not covered are updated by $\frac{0\cdot p_j}{\Pr(o_1)} = \frac{0\cdot 0.08}{0.85} = 0$. Their zero value follows from the fact that, if they were not involved in the test, and the test failed, it is impossible that these statements are faulty. After applying test t_2, no failure occurs. The probabilities of the covered statements which are not already 0 are then updated by $\frac{h_{2,j}\cdot\Pr(d_j|o_1)}{\Pr(o_2)} = \frac{0\cdot 0.09}{0.18} = 0$ and the untouched statements (c_6, c_8) by $\frac{1\cdot\Pr(d_j|o_1)}{\Pr(o_2)} = \frac{1\cdot 0.09}{0.18} = 0.5$. Finally t_3 is applied, which fails. The only covered component with non-zero probability is c_6, and it is updated by $\frac{1\cdot\Pr(d_6|o_2,o_1)}{\Pr(o_3)} = \frac{1\cdot 0.5}{0.5} = 1$ The remaining tests have no influence on the diagnosis.

4 Theoretical Evaluation

In this section, we analyze the performance gain of Bayesian diagnosis with prior h_j knowledge (we will refer to it as H-Bayes) on synthetically generated coverage matrices where the pass/fail outcome of each test is computed using the component intermittency model according to Equation 7. The motivation for using synthetic data next to real-world data is the ability to study the effect of the various parameters in a controlled setting, whereas real programs only represent a few parameter settings in the multi-dimensional parameter space. As explained in Section 3, performance of the fault localization process is measured in terms of the progress and final value of the normalized C_d function. We will consider two reference values for comparison: the number of tests required to reach $C_d = 0.1$ ($T_{0.1}$), and the final quality of the diagnosis $C_d(N)$.

We compare H-Bayes with the classical fault diagnosis techniques presented in Section 3 (Tarantula and Ochiai), and with Single-fault Bayesian diagnosis (intermittency estimated *a posteriori* with Equation 8), for random uniform matrices ($N = 500$ tests, $M = 50$ components, coverage density $\rho = 0.6$, 1000 runs per average). We will refer to it as SF-Bayes.

4.1 Perfect Testability Estimations

First, we analyze the performance of H-Bayes in an environment where the input parameters of intermittency $\hat{h}_j$ for the diagnosis correspond to the actual intermittency values, h_j, of the faults in the system. The results obtained in this subsection are, therefore, regarded as upper bounds of diagnostic performance for H-Bayes. The plot in Figure 1 shows the progress of C_d for each technique, when the h_j values of the simulated faults are drawn from a uniform distribution $U(0, 1)$. It can be seen how H-Bayes provides the best performance over all techniques, by exploiting an accurate intermittency value from the start. With test effort reduction as our goal, H-Bayes reaches $T_{0.1}$ after 8 tests, whereas

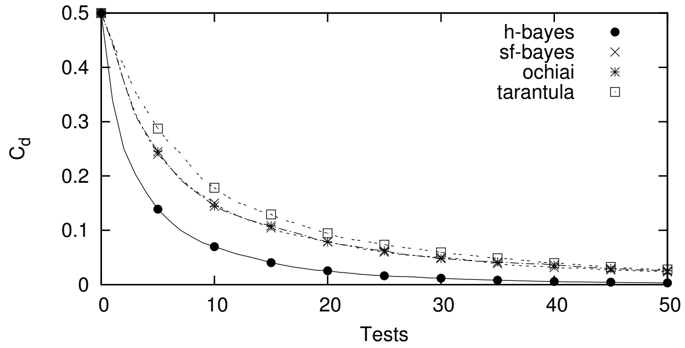

Fig. 1. Performance comparison for synthetic matrices.

the next best technique, SF-Bayes, only reaches this diagnostic quality after 16 tests, which represents a 50% test effort reduction. In this experiment, all techniques are able reach $C_d = 0$ given enough tests. However, H-Bayes reaches a perfect diagnosis after approximately 230 tests, whereas SF-Bayes and the other techniques require the maximum of 500 tests used in the experiment.

4.2 Insufficient Testability Estimations

In a second experiment, we analyze the diagnostic performance penalty of H-Bayes when the estimated intermittency values, $\hat{h}_j$, are sub-optimal, and deviate from the actual values. The actual intermittency value h_j is unknown and must be obtained through *a priori* intermittency estimation, e.g., by means of testability studies, producing sub-optimal results. The plot in Figure 2 shows the effect of noise on the diagnostic performance of H-Bayes for the same synthetic matrices used above. As a reference, the the results of the next best technique (SF-Bayes) are also plotted. The $\hat{h}_j$ values provided to H-Bayes were altered with random noise from the actual h_j, causing deviations of 5%, 10% and up to 25% from the real intermittency values (σ in Figure 2).

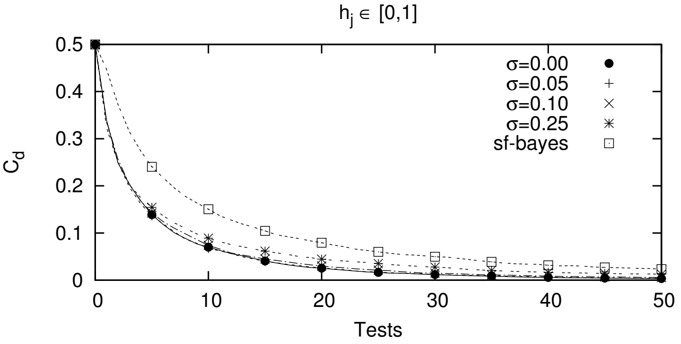

Fig. 2. Effects of erroneous $\hat{h}_j$ estimations.

It can be seen how, on systems with uniformly distributed h_j, the average effect of erroneous $\hat{h}_j$ is negligible, even when intermittency estimations have $\sigma = 0.25$ error. The progress and final diagnostic quality of H-Bayes are still better than those of SF-Bayes.

It is important to note that the average number of tests required to trigger the first failure is related to the expected value of a geometric distribution $X \sim \text{Geo}(p = 1 - h_j)$. Following this, the expected average number of tests which cover the fault, until observing the first failure, is $E[X] = 1/(1 - h_j)$. If a suspicious component c_j is covered more than $1/(1 - h_j)$ times without observing any failure, its probability of being faulty will become close to 0.

As the error in the expected number of tests grows geometrically with $1 - h_j$, diagnostic performance can be severely affected if h_j is close to 1 and its estimation, $\hat{h}_j$, is not accurate. For example, a component with estimated $\hat{h}_j = 0.90$ will be exonerated approximately after 10 passed tests. If the actual intermittency value is $h_j = 0.95$, one would need 20 tests until observing the first failure. This error is even greater for a component with extremely low testability.

5 Empirical Evaluation

This section presents the performance results of H-Bayes when applied to the programs of the well-known Siemens benchmark set [19] and the SIR program `space`.

The Siemens set comprises seven programs, providing one correct version, and a set of faulty versions for each program. Every faulty version contains exactly one fault. For each program, a test suite is provided to achieve full statement and branch coverage. The test suite of the program `space` was sampled to 1,000 out of the original 13,000 test cases to speed up testability calculations. As each program studied comes with a correct version, we use this as test oracle. Table 2 summarizes the programs used for empirical evaluation.

Program Name	# Faults	# Mutants	# LOC	# Tests (N)	Program Type
`print_tokens`	4	491	539	4,130	Lexical Analyzer
`print_tokens2`	9	294	489	4,115	Lexical Analyzer
`replace`	23	757	507	5,542	Pattern Recognition
`schedule`	7	281	397	2,650	Priority Scheduler
`schedule2`	9	212	299	2,710	Priority Scheduler
`tcas`	35	208	174	1,608	Altitude Separation
`tot_info`	19	396	398	1,052	Information Measure
`space`	28	3265	9,564	1,000	ADL Interpreter

Table 2. Summary of the programs used for empirical evaluation.

5.1 Estimation of Diagnosis Parameters

Following existing literature on fault diagnosis, we assume uniform *a priori* fault probability p_j [5]. The values of p_j have little influence on the performance of the diagnosis, as they are "adapted" by the subsequent Bayesian update process.

W.r.t. the estimation of the intermittency value h_j at statement and function point level, the following variation of the PIE testability analysis [6] was used. First, for each component (statement or function) for which we wish to calculate $\hat{h}_j$, a set of mutants $\mathcal{M}_j$ was created. This was done by applying a small set of mutation operators [20] to the arithmetic, logic and indexing operations contained in the statement or function. The mutations were done at the level of the bytecode representation used by the Zoltar [21] fault diagnosis tool. Second, the program's full test suite was run, recording test coverage and failure information for each of the mutants. Finally, the intermittency of each component c_j was obtained by averaging the intermittency of each mutation in $\mathcal{M}_j$, calculated as the ratio of the number of tests which covered the fault but did not produce a failure, to the number of tests which covered the fault, given by

$$\hat{h}_j = \frac{1}{|\mathcal{M}_j|} \cdot \sum_{m \in \mathcal{M}_j} \frac{n_{10}(j)}{n_{10}(j) + n_{11}(j)} \tag{9}$$

This way of estimating h_j is similar to the one used in SF-Bayes *a posteriori*. However, here, we aim at estimating the average intermittency of any fault that *might* occur in a given component, instead of the intermittency of the actual fault being diagnosed.

It is important to note that program mutation can produce equivalent mutants, i.e., altered programs but tested as being correct. These should be removed from the average in Equation 9. In our study, though, we kept those mutants, because the fact that a mutant did not produce any failure in the tests could also mean that there is no test case to render the mutant faulty. Therefore, the values obtained by our intermittency estimation should be seen as pessimistic. This approach was also used in [7].

5.2 Discussion of the Results

This sub-section summarizes the observations on the performance of H-Bayes for diagnosing faults which inserted through the program mutations described above. For the Siemens set, every program statement represents a (potentially faulty) component, for the `space` program it is every function point. 10 random test suites of 500 tests (enough to provide an accurate diagnosis) were created for each program, and executed for each of the mutants.

The plot in Figure 3 shows the progress of the diagnosis cost C_d for all programs. Because we are interested in the initial gain in C_d, only the first one hundred tests applied are displayed. In addition, Table 3 shows, per program, the average number of tests needed to reach (if possible) a $C_d = 0.1$ ($T_{0.1}$), and the C_d after all the 500 tests have been applied. The empirical evaluation confirms the theoretical evaluation. From the outcome it is apparent how the introduction of *a priori* intermittency information greatly improves the performance in the diagnosis, and how it reduces the number of tests substantially, which are required to reach confidence in the diagnosis.

Including *a priori* intermittency information leads to 75% (`replace`) and 73% (`schedule`) cost reduction for SF-Bayes diagnosis and statement component granularity, and 40% cost reduction for the much larger `space` program with function point component granularity.

In the reduction of the test effort in order to reach a preset diagnostic accuracy or confidence, two factors play an important role. First, all diagnosis techniques except H-Bayes are heavily dependent on at least one failing test in order for them to begin with meaningful diagnosis. Otherwise all likelihoods are 0. H-Bayes can perform diagnosis only based on passed tests.

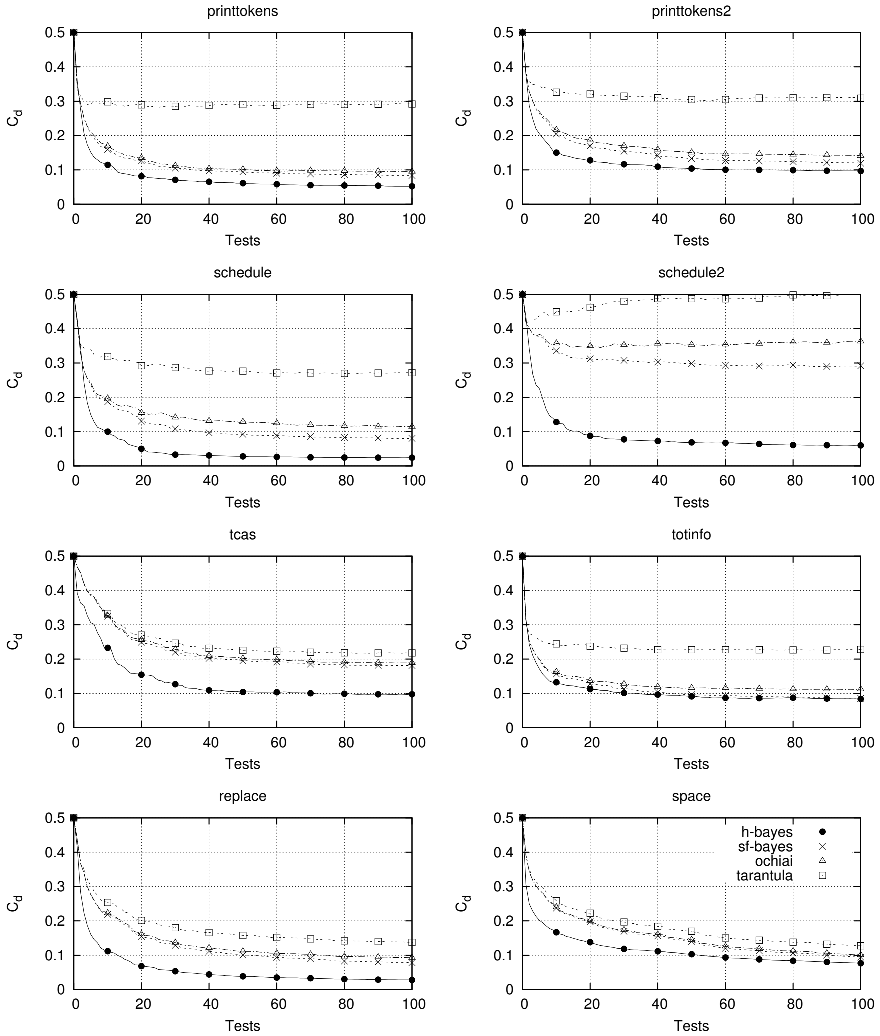

Fig. 3. Performance comparison for mutated programs.

	H-Bayes		SF-Bayes		Ochiai		Tarantula	
program	$T_{0.1}$	$C_d(N)$	$T_{0.1}$	$C_d(N)$	$T_{0.1}$	$C_d(N)$	$T_{0.1}$	$C_d(N)$
print_tokens	13	0.05	36	0.08	53	0.09	-	0.29
print_tokens2	61	0.10	-	0.12	-	0.14	-	0.31
replace	13	0.03	52	0.08	76	0.09	-	0.14
schedule	10	0.02	38	0.08	-	0.12	-	0.27
schedule2	17	0.06	-	0.29	-	0.36	-	0.50
tcas	71	0.10	-	0.18	-	0.19	-	0.22
tot_info	34	0.08	44	0.09	-	0.11	-	0.23
space*	53	0.08	89	0.09	104	0.10	-	0.13

*granularity is at the function level.

Table 3. Detailed results for real programs with mutation faults.

Secondly, exoneration based on passed tests is very conservative for all methods except H-Bayes. If a component has been involved in a large number of test runs that failed, but the component is not faulty, it will take many passed runs to decrease its likelihood of being faulty (l_j). However, when using prior health information represented through *a priori* intermittency, it will take only $1/(1-h_j)$ tests to exonerate the component.

Improvements on diagnosis quality, i.e., reduction of wasted effort for unnecessarily analyzing potentially faulty components, range from 80% for schedule2 and 75% for schedule, to 11% for tot_info and space, for the 500 observations (tests) carried out. The reason for the dramatic improvement of the quality of the diagnosis, especially in the case of schedule2, is the fact that H-Bayes is less affected by ambiguities in the diagnosis, meaning there are different diagnoses with equal likelihoods. If statements, and, therefore, components are part of a sequential block, they will always be used in sequence, and their columns in A (execution patterns) will be identical. Similarity coefficients and SF-Bayes will, therefore, estimate identical likelihood values for all statements with identical execution patterns, making the diagnosis ambiguous, leading to an increase in diagnosis effort. However, the testability values estimated for those are not necessarily identical. Therefore, even if their execution patterns are identical, H-Bayes will rank first the components whose h_j is closer to the intermittency of failures, effectively alleviating ambiguity.

5.3 Threats to Validity

It must be noted that the validity of the results presented is threatened by the fact that the same set of mutants were used to estimate h_j *and* to perform the diagnosis. This could produce too optimistic results, as the estimated $\hat{h}_j$ corresponds exactly to the average of the actual h_j of the mutants being diagnosed. This sub-section presents an experiment with real faults found during the development of the space program in order to assess this threat. We used space because of the function point component granularity, resulting in a fixed component topology when statements are amended. That way, we can use the same existing *a priori* testability or intermittency estimate. Adding or removing components would render data gathered from previous diagnoses useless, because of a changed system topology.

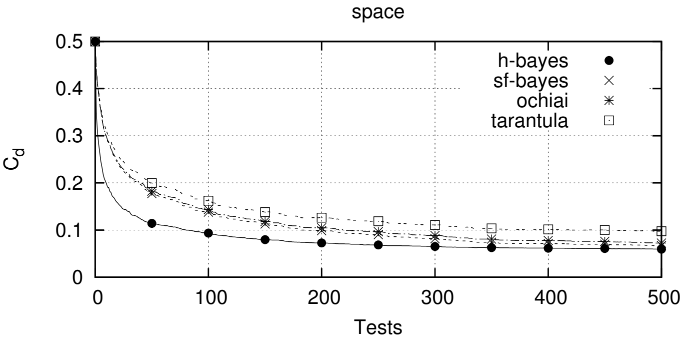

Fig. 4. Diagnosis performance for `space`.

	H-Bayes		SF-Bayes		Ochiai		Tarantula	
program	$T_{0.1}$	$C_d(N)$	$T_{0.1}$	$C_d(N)$	$T_{0.1}$	$C_d(N)$	$T_{0.1}$	$C_d(N)$
space	81	0.09	182	0.14	219	0.14	421	0.16

Table 4. Detailed results for the real faults of `space`.

The plot in Figure 4, and the values in Table 4 show the results of this experiment. By using H-Bayes, residual diagnostic effort is reduced by 30% compared to SF-Bayes, and test effort is reduced by 55%.

Given the dramatic improvements achievable by incorporating testability and thus intermittency information as parameter to the diagnosis process, a final note regarding the cost of obtaining such information is obligatory. The cost of a testability study is not prohibitive in terms of algorithmic complexity, as it scales with $O(k \cdot M \cdot N)$, where k is the average number of mutants generated per component, and M, N are the number of mutable components and tests, respectively. However, it is nevertheless, expensive in terms of execution time. For example, the testability study for `space` involved executing 1000 tests for more than 3200 mutants, which took more than 2 days on a 2.33GHz Xeon Linux server. The calculations can be sped up using multiple CPUs to run multiple mutants simultaneously.

6 Related Work

Automated fault localization techniques minimize diagnostic cost and support debugging when failures occur during software testing. Statistical approaches include [1, 2, 22–25]. A recent, probabilistic approach of acceptable complexity is [5]. Although, different in the way they derive the ranking of the diagnosis, all techniques are based on measuring the coverage information and failure pattern of a program (also termed its spectrum), while ignoring or deriving the intermittency of the faults. The novelty of our approach consists in factoring the intermittency estimation problem out of the diagnostic problem, so that intermittency can be exploited from the beginning. Fault intermittency in software is related to testability, in that testability represents *the degree to which software reveals faults during testing* [12]. Testability quantification has been approached at the class level [26], function level [13] and statement level [6, 14–17]. However, of the current state of the art, only [6] allows for a straightforward usage as probabilities in our experiments.

7 Conclusions and Future Work

Fault intermittency modeling is an essential problem of fault localization. Previous work in diagnosis has considered intermittency modeling a part of diagnosis process. In this paper we have studied the improvement of the software fault localization process in the case that intermittency represented through testability information is available *a priori*. Our results indicate that testability information can reduce the number of tests required for an acceptable diagnosis significantly, both in theoretical cases as well as in realistic cases. H-Bayes reduces the average number of tests required to reach the same confidence in the diagnosis as the next best technique by 55%, and the average effort for coming to the diagnosis by 30% for real programs with real faults. An additional observation from our experiments is that H-Bayes is relatively robust w.r.t. inaccurate intermittency estimates.

The improvement in the fault localization process was traded against an expensive testability analysis, although, it should be noted that, in practice, the cost of such analysis is amortized over many debug-repair cycles. Future work will investigate the usage of alternative testability estimation techniques, e.g. bridging the gap between static techniques (DRR) and the probabilistic approach needed for H-Bayes. As the estimation error of intermittency may affect the quality of the diagnosis, methods to adapt or correct h_j as soon as additional information becomes available are also relevant for further research.

References

1. Abreu, R., Zoeteweij, P., van Gemund, A.: On the accuracy of spectrum-based fault localization. In: Proc. TAIC PART'07
2. Jones, J.A., Harrold, M.J., Stasko, J.: Visualization of test information to assist fault localization. In: Proc. ICSE'02
3. Abreu, R., Zoeteweij, P., van Gemund, A.J.: A new bayesian approach to multiple intermittent fault diagnosis. In: Proc. IJCAI'09
4. de Kleer, J.: Diagnosing multiple persistent and intermittent faults. In: Proc. IJCAI'09

5. Abreu, R., Zoeteweij, P., van Gemund, A.J.: Spectrum-based multiple fault localization. In: Proc. ASE'09
6. Voas, J.M.: Pie: A dynamic failure-based technique. IEEE Transactions in Software Engineering **18**(8) (1992) 717–727
7. Rothermel, G., Untch, R.H., Chu, C., Harrold, M.J.: Prioritizing test cases for regression testing. IEEE TSE **27**(10) (2001)
8. de Kleer, J., Price, B., Kuhn, L., Do, M., Zhou, R.: A framework for continuously estimating persistent and intermittent failure probabilities. In: Proc. DX'08
9. Raghavan, V., Shakeri, M., Pattipati, K.R.: Test sequencing algorithms with unreliable tests. IEEE TSMC **29**(4) (1999)
10. Gonzalez, A., Piel, E., Gross, H.G., van Gemund, A.J.: Prioritizing tests for software fault localization. In: Proc. QSIC'10. (2010) (To appear).
11. Gonzalez, A., Abreu, R., Gross, H.G., van Gemund, A.J.: A diagnostic approach to test prioritization (technical report). Technical report, Softw. Eng. Grp., Delft University of Technology (2010) http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2010-007.pdf.
12. Voas, J.M., Miller, K.W.: Software testability: The new verification. IEEE Software **12**(3) (1995) 17–28
13. Freedman, R.S.: Testability of software components. IEEE Transactions in Software Engineering **17**(6) (1991) 553–564
14. Voas, J.M., Miller, K.W.: Semantic metrics for software testability. Journal of Systems and Software **20**(3) (1993) 207–216
15. Woodward, M.R., Al-Khanjari, Z.A.: Testability, fault size and the domain-to-range ratio: An eternal triangle. SIGSOFT Software Engineering Notes **25**(5) (2000) 168–172
16. Zhao, L.: A new approach for software testability analysis. In: ICSE '06, New York, NY, USA, ACM (2006) 985–988
17. Offutt, A.J., Hayes, J.H.: A semantic model of program faults. SIGSOFT Software Engineering Notes **21**(3) (1996) 195–200
18. Jiang, B., Zhang, Z., Chan, W., Tse, T.: Adaptive random test case prioritization. In: Proc. ASE'09
19. Hutchins, M., Foster, H., Goradia, T., Ostrand, T.: Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In: Proc. ICSE '94
20. Offutt, A.J., Lee, A., Rothermel, G., Untch, R.H., Zapf, C.: An experimental determination of sufficient mutant operators. ACM Trans. Softw. Eng. Methodol. **5** (April 1996) 99–118
21. Janssen, T., Abreu, R., van Gemund, A.J.C.: Zoltar: A toolset for automatic fault localization. In: Proc. ASE'09 - Tool Demonstrations
22. Liblit, B.: Cooperative debugging with five hundred million test cases. In: Proc. ISSTA'08
23. Liu, C., Yan, X., Fei, L., Han, J., Midkiff, S.P.: Sober: Statistical model-based bug localization. In: Proc. ESEC/FSE-13
24. Renieris, M., Reiss, S.P.: Fault localization with nearest neighbor queries. In: Proc. ASE'03
25. Wong, W., Wei, T., Qi, Y., Zhao, L.: A crosstab-based statistical method for effective fault localization. In: Proc. ICST'08
26. Bruntink, M., van Deursen, A.: An empirical study into class testability. J. Syst. Softw. **79**(9) (2006) 1219–1232

TUD-SERG-2010-011
ISSN 1872-5392