Spectrum-based Multiple Fault Localization*

Rui Abreu and Peter Zoeteweij and Arjan J.C. van Gemund
Embedded Software Lab
Faculty Electrical Eng., Mathematics, and Computer Science
Delft University of Technology
The Netherlands
Email: {r.f.abreu, p.zoeteweij, a.j.c.vangemund}@tudelft.nl

***Abstract*—Fault diagnosis approaches can generally be categorized into spectrum-based fault localization (SFL, correlating failures with abstractions of program traces), and model-based diagnosis (MBD, logic reasoning over a behavioral model). Although MBD approaches are inherently more accurate than SFL, their high computational complexity prohibits application to large programs. We present a framework to combine the best of both worlds, coined BARINEL. The program is modeled using abstractions of program traces (as in SFL) while Bayesian reasoning is used to deduce multiple-fault candidates and their probabilities (as in MBD). A particular feature of BARINEL is the usage of a probabilistic component model that accounts for the fact that faulty components may fail intermittently. Experimental results on both synthetic and real software programs show that BARINEL typically outperforms current SFL approaches at a cost complexity that is only marginally higher. In the context of single faults this superiority is established by formal proof.**

***Keywords*-Software fault diagnosis, program spectra, statistical and reasoning approaches.**

I. INTRODUCTION

Automatic fault localization techniques aid developers/testers to pinpoint the root cause of software failures, thereby reducing the debugging effort. Two major approaches can be distinguished, (1) *spectrum-based fault localization* (SFL), and (2) *model-based diagnosis* or *debugging* (MBD). SFL uses abstraction of program traces to correlate software component activity with program failures (a *statistical* approach) [4], [14], [20], [24], [29], [37]. Although statistical approaches are very attractive from complexity point of view, there is no reasoning in terms of *multiple* faults to explain all failures. However, with current defect densities and program sizes multiple faults are a fact of life. Inherently taking a single-fault approach, SFL performs less when failures are caused by different faults as no specific correlation between a failure and a fault can be established.

MBD approaches deduce component failure through *logic reasoning* [9], [11], [12], [26], [28], [35] using propositional models of component behavior. An inherent, strong point of MBD is that it reasons in terms of multiple faults. In contrast to SFL's simple component ranking, its diagnostic report contains multiple-fault candidates, providing more diagnostic information compared to the one-dimensional list in SFL. Ranking is determined in terms of (multiple) fault *probability*, a more solid basis for candidate ranking than statistical similarity. While inherently more accurate than statistical approaches, the main disadvantages of reasoning approaches are (1) the need for model generation, usually with the help of static analysis that is unable to capture dynamic data dependencies / conditional control flow, and (2) the exponential cost of diagnosis candidates generation, typically prohibiting its use for programs larger than a few hundred lines [26].

Aimed to combine the best of both worlds, in this paper we present a novel, Bayesian reasoning approach to spectrum-based multiple fault localization. Similar to SFL, we model program behavior in terms of program spectra, abstracting from modeling specific components and data dependencies. Similar to MBD, we employ a probabilistic (Bayesian) approach to deduce multiple-fault candidates and their probabilities, yielding an information-rich diagnostic ranking. To solve the inherent exponential complexity problem in MBD, we use a novel, heuristic approach to generate the most significant diagnosis candidates only, dramatically reducing computational complexity. As a result, our approach can be used on large, real-world programs without any problem.

A central feature of our contribution is the use of a generic, probabilistic component failure model that accounts for the fact that a faulty component j may still behave as expected (with *health* probability h_j), i.e., need not contribute to a program failure (aka intermittent fault behavior). Such an intermittency model is crucial for MBD approaches where (deterministic) component behavior is abstracted to a modeling level where particular input and output values are mapped to, e.g., ranges, as shown in [3], [8]. Our diagnosis approach is based on computing h_j using a maximum likelihood estimation procedure, optimally exploiting all information contained by the program spectra. The result is an improved probability computation for each diagnostic

*This work has been carried out as part of the TRADER project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the BSIK03021 program.

candidate, thus improving ranking quality. As for large systems the number of diagnostic candidates is extremely large (the actual faults being amongst them), increased ranking quality is a critical to diagnostic performance.

Results on synthetic program models show that our approach provides better diagnostic performance than all similarity coefficients known to date in SFL, as well as other spectrum-based MBD approaches (including our previous contribution [3]). This confirms that Bayesian reasoning inherently delivers better diagnostic performance than similarity-based ranking. Similarly, results on real programs demonstrate that our approach is equal or better than all SFL approaches at a time and space complexity that is only marginally higher than SFL.

In particular, the paper makes the following contributions

- We present our new approach for the candidate probability computation which features a maximum likelihood estimation algorithm to compute the h_j of all components involved in the diagnosis. The approach is coined BARINEL[1], which is the name of the software implementation of our method;
- We study the inherent performance properties of our approach using synthetic program spectra based on multiple injected faults of which the h_j are given, and compare the accuracy with statistical approaches, as well as previous spectrum-based reasoning work;
- We prove that for the single-fault case our approach is optimal. We empirically demonstrate this result by comparing diagnostic accuracy of our approach with a large body of existing results for the (single-fault) Siemens benchmark suite;
- We compare BARINEL with existing work (Tarantula, Ochiai, and a previous, approximate Bayesian approach) for a set of programs commonly used, extended with multiple faults, demonstrating the improved performance of our approach and the low computation complexity involved.

To the best of our knowledge, our Bayesian approach to spectrum-based fault localization has not been described before. The paper is organized as follows. In the next section we present the current approach to SFL and illustrate why a reasoning approach can improve diagnostic performance. In Section III we present our BARINEL approach to fault localization. In Section IV, the approach is theoretically evaluated, while in Section V real programs are used to assess the capabilities of our technique. We compare BARINEL with related work in Section VI. In Section VII we conclude and discuss future work.

[1]BARINEL stands for Bayesian AppRoach to dIagnose iNtErmittent fauLts. A barinel is a type of caravel used by the Portuguese sailors during their discoveries.

$$\begin{array}{c} \\ N \text{ spectra} \end{array} \begin{array}{c} \overbrace{\qquad\qquad\qquad}^{M \text{ components}} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{M2} & \dots & a_{NM} \end{bmatrix} \end{array} \begin{array}{c} \text{error} \\ \text{detection} \\ \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} \end{array}$$

Figure 1. Input to SFL

II. CURRENT SFL APPROACH

In the following we summarize the traditional, statistical approach to spectrum-based fault localization. A program under analysis comprises a set of M components (e.g., functions, statements) c_j where $j \in \{1, \dots, M\}$, and can have multiple faults, the number being denoted C (fault cardinality). A *diagnostic report* $D =< \dots, d_k, \dots >$ is an ordered set of diagnostic (possible multiple-fault) candidates d_k ordered in terms of likelihood to be the true diagnosis. Statistical approaches yield a single-fault diagnostic report with the M components ordered in terms of statistical similarity (e.g., $< \{3\}, \{1\}, \dots >$, in terms of the indices j of the components c_j).

Program (component) activity is recorded in terms of program spectra [15]. This data is collected at run-time, and typically consists of a number of counters or flags for the different components of a program. In this paper we use the so-called hit spectra, which indicate whether a component was involved in a (test) run or not.

Both spectra and program pass/fail (test) information is input to SFL (see Figure 1). The program spectra are expressed in terms of the $N \times M$ *activity matrix* A. An element a_{ij} is equal to 1 if component j was observed to be involved in the execution of run i, and 0 otherwise. For $j \leq M$, the row A_{i*} indicates whether a component was executed in run i, whereas the column A_{*j} indicates in which runs component j was involved. The pass/fail information is stored in a vector e, the *error vector*, where e_i signifies whether run i has *passed* ($e_i = 0$) or *failed* ($e_i = 1$). Note that the pair (A, e) is the only input to SFL.

In SFL one measures the statistical similarity between the error vector e and the activity profile column A_{*j} for each component c_j. This similarity is quantified by a *similarity coefficient*, expressed in terms of four counters $n_{pq}(j)$ that count the number of elements in which A_{*j} and e contain respective values p and q, i.e, for $p, q \in \{0, 1\}$, we define

$$n_{pq}(j) = |\{i \mid a_{ij} = p \wedge e_i = q\}|$$

An example of a well-known similarity coefficient is the Ochiai coefficient [4]

$$s_O(j) = \frac{n_{11}(j)}{\sqrt{(n_{11}(j) + n_{10}(j)) \cdot (n_{11}(j) + n_{01}(j))}} \quad (1)$$

To illustrate how SFL works, consider the following toy program:

```
if (a == 1)
   y = f1(x);
if (b == 1)
   y = f2(x);
if (x == 0)
   y = f3(y);
```

where the three functions are given by

```
f1(x){ return x * 100; // c1; fault: should be 10}
f2(x){ return x / 100; // c2; fault: should be 10}
f3(y){ return y + 10;  // c3; correct code}
```

Consider a function spectrum based on the three function components `c1`, `c2`, and `c3`. Suppose, we have 5 test runs with inputs `(a b x)` equal to `(1 0 0)`, `(0 1 0)`, `(1 0 1)`, `(0 1 1)`, `(1 1 1)`, respectively. The spectra (A) and pass/fail information (e) are given in Table I. As can be seen, the

	c_1	c_2	c_3	e
	1	0	1	1
	0	1	1	1
	1	0	0	1
	0	1	0	1
	1	1	0	0
$n_{11}(j)$	2	2	2	
$n_{10}(j)$	1	1	0	
$n_{01}(j)$	2	2	2	
$s_O(j)$	0.6	0.6	0.7	

Table I
(A, e) OF TOY PROGRAM, INCLUDING HIT COUNTERS AND OCHIAI SIMILARITY COEFFICIENTS.

Ochiai coefficient (and all other similarity coefficients such as Tarantula [20] and Jaccard [4]) rank c_3 highest although c_1, c_2 are actually at fault. This is due to the fact that the statistical approaches do not *reason* over (A, e) in terms of a behavioral model of the program.

III. SPECTRUM-BASED REASONING

In this section we describe our BARINEL approach. If one considers the above spectrum (and error vector), a number of facts are evident. From the third row it is obvious that c_1 must be among the faulty components, as the failure can only be caused by at least one faulty component. From the fourth row it follows that c_2 must also be faulty. In contrast, from third and fourth rows it also follows that c_3 can never be a single fault. In addition, c_3 can also not be part of a double fault such as $\{1, 3\}$ due to the fourth row, nor $\{2, 3\}$ due to the third row. Although the last row partially exonerates c_1 and c_2 in the statistical technique (through n_{10} in Eq. (1)), in a reasoning approach this row cannot change the only possible outcome: $\{1, 2\}$ is the only possible diagnosis. Note, that we do not have to model the program in great detail (e.g., at the statement level, such as in [26]). Only the dynamic observations available at the spectral level have served as the basis for our above reasoning. In the following we describe our spectrum-based reasoning approach in which we apply MBD principles based on the observations from (A, e).

A. Candidate Generation

Model-based reasoning approaches yield a diagnostic report that comprise multiple-fault candidates d_k ordered in terms of probability (e.g., $< \{4\}, \{1, 3\}, \ldots >$, meaning that either component c_4 is at fault, or components c_1 *and* c_3 are at fault, etc.). As in any MBD approach we base ourselves on a model of the program. Unlike many MBD approaches, however, we refrain from detailed modeling, e.g., at the statement level, but assume a generic component model (actually, in this paper, we will model statements as components). Each component (c_j) is modeled in terms of the logical proposition

$$h_j \Rightarrow (ok_{inp_j} \Rightarrow ok_{out_j}) \tag{2}$$

where the booleans h_j, ok_{inp_j}, and ok_{out_j} model component health, and the (value) correctness of the component's input and output variables, respectively. The above model specifies nominal (required) behavior: when the component is correct and its inputs are correct, then the outputs must be correct. Note that the above model does not specify faulty behavior. Even when the component is faulty and/or the input values are incorrect it is still possible that the component delivers a correct output. Hence, a program pass does not imply correctness of the components involved (the last row in the example (A, e) does not logically exonerate c_1 nor c_2).

By instantiating the above equation for each component involved in a particular run (row in A) a set of logical propositions is formed. Since the input variables of each test can be assumed to be correct, and since the output correctness of the final component in the invocation chain is given by e (pass implies correct, fail implies incorrect), we can logically infer component health information from each row in (A, e). For the above example we directly obtain the following health propositions for h_j:

$$\begin{aligned} \neg h_1 \vee \neg h_3 \quad & (c_1 \text{ and/or } c_3 \text{ faulty}) \\ \neg h_2 \vee \neg h_3 \quad & (c_2 \text{ and/or } c_3 \text{ faulty}) \\ \neg h_1 \quad & (c_1 \text{ faulty}) \\ \neg h_2 \quad & (c_2 \text{ faulty}) \end{aligned}$$

The above health propositions have a direct correspondence with the original matrix structure. Note that only failing runs lead to a corresponding health propositions, as, due to the conservative component model, from a passing run no additional health information can be inferred.

As in most MBD approaches, the above health propositions are subsequently combined to a diagnosis by computing the so-called minimal hitting sets (MHS, aka minimal set

cover [7]), i.e., the minimal health propositions that cover the above propositions. In the above case, there is only one MHS, given by

$$\neg h_1 \wedge \neg h_2 \quad (c_1 \text{ and } c_2 \text{ faulty})$$

that covers all four previous health propositions. Thus $D =< \{1, 2\} >$ comprises only 1 (double-fault) candidate, which therefore is the actual diagnosis.

In summary, our spectrum-based model-based reasoning approach uses a generic component model to compile (A, e) into corresponding health propositions, which are subsequently transformed into D using an MHS algorithm. The latter step is generally responsible for the prohibitive cost of reasoning approaches. However, in our approach we use an ultra-low-cost heuristic MHS algorithm called STACCATO (STAtistiCs-direCted minimAl hiTing set algOrithm) [2] to extract only the significant set of multiple-fault candidates d_k, avoiding the needless generation of a possibly exponential number of diagnostic candidates. This feature allows our Bayesian reasoning approach to be applied to real-world programs without any problem as shown later on.

B. Candidate Ranking

As mentioned earlier, unlike statistical approaches which return all M component indices, model-based reasoning approaches only return diagnosis candidates d_k that are logically consistent with the observations. Despite this candidate reduction, the number of remaining candidates d_k is typically large, and not all of them are equally probable. Hence, the computation of diagnosis candidate probabilities $\Pr(d_k)$ to establish a *ranking* is critical to the diagnostic performance of model-based reasoning approaches. In MBD the probability that a diagnosis candidate is the actual diagnosis is computed using Bayes' rule, that updates the probability of a particular candidate d_k given new observational evidence (from a new program run).

The Bayesian probability update, in fact, can be seen as the foundation for the derivation of diagnostic candidates in any reasoning approach, i.e., (1) deducing whether a candidate diagnosis d_k is consistent with the observations, and (2) computing the posterior probability $\Pr(d_k)$ of that candidate being the actual diagnosis. Rather than computing $\Pr(d_k)$ for *all* possible candidates, just to find that most of them have $\Pr(d_k) = 0$, candidate generation algorithms are used as shown before, but the Bayesian reasoning framework remains the formal basis.

For each diagnosis candidate d_k the probability that it describes the actual system fault state depends on the extent to which d_k explains all observations. To compute the posterior probability that d_k is the true diagnosis given observation obs_i (obs_i refers to the coverage and error info for test i) Bayes' rule is used:

$$\Pr(d_k|obs_i) = \frac{\Pr(obs_i|d_k)}{\Pr(obs_i)} \cdot \Pr(d_k|obs_{i-1}) \tag{3}$$

The denominator $\Pr(obs_i)$ is a normalizing term that is identical for all d_k and thus needs not be computed directly. $\Pr(d_k|obs_{i-1})$ is the prior probability of d_k. In absence of any observation, $\Pr(d_k|obs_{i-1})$ defaults to $\Pr(d_k) = p^{|d_k|} \cdot (1-p)^{M-|d_k|}$, where p denotes the *a priori* probability that component c_j is at fault, which in practice we set to $p_j = p$. $\Pr(obs_i|d_k)$ is defined as

$$\Pr(obs_i|d_k) = \begin{cases} 0 & \text{if } obs_i \wedge d_k \text{ are inconsistent;} \\ 1 & \text{if } obs_i \text{ is unique to } d_k; \\ \varepsilon & \text{otherwise.} \end{cases} \tag{4}$$

As mentioned earlier, rather than updating each candidate only candidates derived from the candidate generation algorithm are updated, implying that the 0-clause need not be considered in practice.

In model-based reasoning, many policies exist for ε [8]. A well-known, intuitive policy is to approximate $\varepsilon = 1/\#obs$ where $\#obs$ is the number of observations that can be explained by diagnosis d_k (the approximation comes from the fact that not all observations belonging to d_k may be equally likely). Our epsilon policy differs, however, due to our choice to use an *intermittent* component failure model, extending h_j's binary definition to $h_j \in [0, 1]$, where h_j expresses the probability that faulty component j produces correct output ($h_j = 0$ means persistently failing, and $h_j = 1$ essentially means healthy, i.e., never inducing failures). The reasons for the intermittent failure model are as follows.

- In many practical situations faults manifest themselves intermittently. This especially applies to software where faulty components typically deliver a fraction of incorrect outputs when executed repeatedly (with different inputs). Note that this directly relates to the fact that in our model we abstract from actual input and output values as mentioned earlier.
- Although the component model in Eq. (2) does allow a faulty component to exhibit correct behavior, the binary health h_j does not enable us to exploit the information contained in the *number* of passing or failing runs in which the component is involved. The intermittent model enables us to more precisely indict or exonerate a component as more information (runs) are available. Thus the resulting probability ranking can be refined, optimally exploiting the information present in (A, e).

Given the intermittency model, for an observation $obs_i = (A_{i*}, e_i)$, the epsilon policy in Eq. (4) becomes

$$\varepsilon = \begin{cases} \prod_{j \in d_k \wedge a_{ij}=1} h_j & \text{if } e_i = 0 \\ 1 - \prod_{j \in d_k \wedge a_{ij}=1} h_j & \text{if } e_i = 1 \end{cases} \tag{5}$$

Eq. (5) follows from the fact that the probability that a run passes is the product of the probability that each

involved, faulty component exhibits correct behavior (or-model, we assume components fail independently, a standard assumption in fault diagnosis for tractability reasons).

C. Health Probability Estimation

A particular problem with the use of intermittent component models is that the h_j are now real-valued. In traditional approaches, $\Pr(d_k)$ given a set of observations (A, e) is computed by equating the h_j to false or true corresponding to the candidate d_k under study. Now that $\Pr(d_k)$ has become a real-valued function of h_j a new approach is needed. The key idea underlying our approach is that for each candidate d_k we compute the h_j for the candidate's faulty components that *maximizes the probability* $\Pr(e|d_k)$ *of the outcome* e *occurring, conditioned on that candidate* d_k (maximum likelihood estimation for naive Bayes classifier d_k). Hence, h_j is solved by maximizing $\Pr(e|d_k)$ under the above epsilon policy, according to

$$H = \arg\max_{H} \Pr(e|d_k)$$

where $H = \{h_j \mid j \in d_k\}$. For example, suppose we measure the following spectrum (the previous example spectrum yielded a single diagnosis, defeating the purpose of the current illustration):

c_1	c_2	c_3	e
1	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0

Candidate generation yields two double-fault candidates $d_1 = \{1, 2\}$, and d_2 = {1,3}. Consider the computation of $\Pr(d_1)$. From Eq. (4) and Eq. (5) it follows

$$\begin{aligned} \Pr(e_1|d_1) &= 1 - h_1 h_2 \\ \Pr(e_2|d_1) &= 1 - h_2 \\ \Pr(e_3|d_1) &= 1 - h_1 \\ \Pr(e_4|d_1) &= h_1 \end{aligned}$$

As the 4 observations $e_1, \ldots, e_4$ are independent it follows

$$\Pr(e|d_1) = (1 - h_1 \cdot h_2) \cdot (1 - h_2) \cdot (1 - h_1) \cdot h_1 \qquad (6)$$

Assuming candidate d_1 is the actual diagnosis, the corresponding h_j are determined by maximum likelihood estimation, i.e., maximizing Eq. (6). For d_1 it follows that $h_1 = 0.47$ and $h_2 = 0.19$ yielding $\Pr(e|d_1) = 0.185$ (note, that c_2 has much lower health than c_1 as c_2 is not exonerated in the last matrix row, in contrast to c_1). Applying the same procedure for d_2 yields $\Pr(e|d_2) = 0.036$ (with corresponding $h_1 = 0.41$, $h_3 = 0.50$). Assuming both candidates have equal prior probability p^2 (both are double-fault candidates) and applying Eq. (3) it follows $\Pr(d_1|e) = 0.185 \cdot p^2/\Pr(e)$ and $\Pr(d_2|e) = 0.036 \cdot p^2/\Pr(e)$. After normalization it follows $\Pr(d_1|e) = 0.839$ and $\Pr(d_2|e) = 0.161$. Consequently, the ranked diagnosis is given by $D =< \{1, 2\}, \{1, 3\} >$. Thus, apart from c_1, c_2 is much more likely to be at fault than c_3 and debugging would commence with c_1 and c_2.

Algorithm 1 Diagnostic Algorithm: BARINEL

Inputs: Activity matrix A, error vector e,
Output: Diagnostic Report D

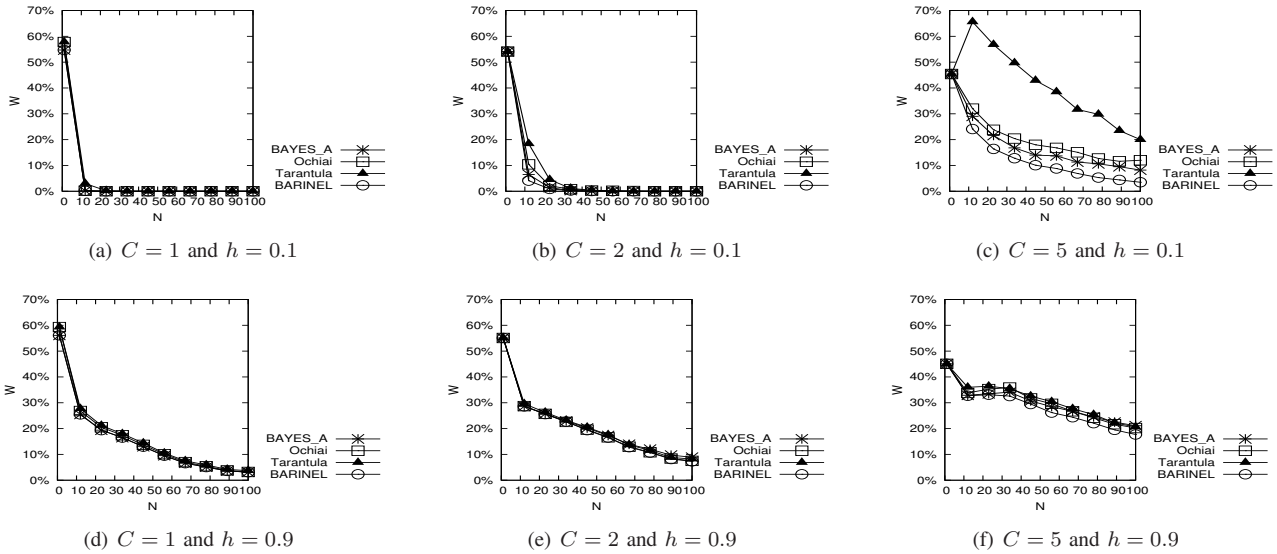
```
1  γ ← ϵ
2  D ← STACCATO((A, e)) ▷ Compute MHS
3  for all d_k ∈ D do
4      expr ← GENERATEPR((A, e), d_k)
5      i ← 0
6      Pr[d_k]^i ← 0
7      repeat
8          i ← i + 1
9          for all j ∈ d_k do
10             g_j ← g_j + γ · ∇expr(g_j)
11         end for
12         Pr[d_k]^i ← EVALUATE(expr, ∀_{j∈d_k} g_j)
13     until |Pr[d_k]^{i−1} − Pr[d_k]^i| ≤ ξ
14 end for
15 return SORT(D, Pr)
```

D. Algorithm

Our spectrum-based reasoning approach is described in Algorithm 1 and comprises three main phases. In the first phase a list of candidates D is computed from (A, e) using STACCATO, a ultra-low cost algorithm. This performance is achieved at the cost of completeness as solutions are truncated at 100 candidates. Nevertheless, experiments [2] have shown that no significant solution was ever missed.

In the second phase $\Pr(d_k)$ is computed for each candidate in D. First, GENERATEPR derives for every candidate d_k the probability $\Pr(e|d_k)$ for the set of observations (A, e). Subsequently, all h_j are computed such that they maximize $\Pr(e|d_k)$. To solve the maximization problem we apply a simple gradient ascent procedure [5] bounded within the domain $0 \leq h_j \leq 1$ (the ∇ operator signifies the gradient computation). As the h_j expressions that need to be maximized are simple, and the domain is bounded to $[0, 1]$, the gradient ascent procedure exhibits reasonably rapid convergence for all M and C (see Section V-C for detailed complexity analysis). Note that the linear convergence of the simple, gradient ascent procedure can be improved to a quadratic convergence (e.g., Newton's method), yielding significant speedup. However, the current implementation already delivers satisfactory performance.

In the third and final phase, for each d_k the diagnoses are ranked according to $\Pr(d_k|(A, e))$, which is computed by EVALUATE based on the usual Bayesian update (Eq. (3) for each row. The algorithm has been implemented within the BARINEL toolset that comprises C program instrumentation (using LLVM [21]) and off-line diagnostic analysis support. The analysis module supports the BARINEL algorithm as well as most statistical methods [16]. Note that our approach is independent of test case ordering.



(a) $C = 1$ and $h = 0.1$ (b) $C = 2$ and $h = 0.1$ (c) $C = 5$ and $h = 0.1$

(d) $C = 1$ and $h = 0.9$ (e) $C = 2$ and $h = 0.9$ (f) $C = 5$ and $h = 0.9$

Figure 2. Wasted effort W vs. N for several settings of C and h

IV. Theoretical Evaluation

In order to assess the diagnostic performance properties of our approach we generate synthetic observations based on random (A, e) generated for various values of N, M, and the number of injected faults C (cardinality). The reason for the synthetic experiments (next to the real programs in the next section) is that we can vary parameters of interest in a controlled setting, whereas real programs typically represent only one particular parameter setting. Component activity a_{ij} is sampled from a Bernoulli distribution with parameter r, i.e., the probability a component is involved in a row of A equals r. For the faulty components c_j (without loss of generality we select the first C components, i.e., $c_1, \ldots, c_C$ are faulty) we set the component healths (intermittency rates) h_j. Thus the probability of a component j being involved *and* generating a failure equals $r \cdot (1 - h_j)$. A row i in A generates an error ($e_i = 1$) if at least 1 of the C components generates a failure (or-model). Measurements for a specific (N, M, C, r, g) scenario are averaged over $1,000$ sample matrices, yielding a coefficient of variance of approximately 0.02.

We compare the accuracy of our Bayesian framework with two state-of-the-art spectrum-based fault localization methods Ochiai and Tarantula, and with a previous Bayesian approach, called Bayes-A [3]. The difference between Barinel and Bayes-A is the epsilon policy. In Bayes-A a simple policy is used that is based on an approximation of the h_j values instead of executing the maximum likelihood estimation procedure (for details see [3]).

Diagnostic performance is measured in terms of a diagnostic performance metric W that measures the percentage of excess *work* incurred in finding all components at fault. The metric is an improvement on metrics typically found in software debugging which measure debugging effort [4], [29]. We use *wasted* effort instead of total effort because in our multiple-fault research context we wish the metric to be independent of the number of faults C in the program to enable an unbiased evaluation of the effect of C on W. Thus, regardless of C, $W = 0$ represents an ideal diagnosis technique (all C faulty components on top of the ranking, no effort wasted on testing other components to find they are not faulty), while $W = 1$ represents the worst case (testing all $M - C$ healthy components until arriving at the C faulty ones). For example, consider a $M = 5$ component program with the following diagnostic report $D =< \{4, 5\}, \{4, 3\}, \{1, 2\} >$, while c_1 and c_2 are actually faulty. The first diagnosis candidate leads the developer to inspect c_4 and c_5. As both components are healthy, W is increased with $\frac{2}{5}$. Using the new information that $h_4 = h_5 = 1.0$ the probabilities of the remaining candidates are updated by rerunning Barinel, leading to $\Pr(\{4, 3\}) = 0$ (c_4 can no longer be part of a multiple fault). Consequently, candidate $\{4, 3\}$ is also discarded, avoiding wasting additional debugging effort. The next components to be inspected are c_1 and c_2. As they are both faulty, no more effort is wasted[2]. Consequently, $W = \frac{2}{5}$.

The graphs in Figure 2 plot W versus N for $M = 20$, $r = 0.6$ (the trends for other M and r values are essentially the same, $r = 0.6$ is typical for the Siemens suite), and different values for C and h (in our experiments we set all $h_j = h$). A number of common properties emerge. All plots show that W for $N = 1$ is similar to r, which agrees with the

[2]Effort, as defined in [4], [29], would be increased by $\frac{2}{5}$ to account for the fact that both components were inspected.

fact that there are on average $(M-C)\cdot r$ components which would have to be inspected in vain. For sufficiently large N all approaches produce an optimal diagnosis, as there are sufficient runs for all approaches to correctly single out the faulty components. For small h_j, W converges quicker than for large h_j as computations involving the faulty components are much more prone to failure, while for large h_j the faulty components behave almost similar to healthy components, requiring more observations (larger N) to stand out in the ranking. Also for larger C more observations are required before the faulty components are isolated. This is due to the fact that failure behavior can be caused by much more components, reducing the correlation between failure and particular component involvement.

The plots confirm that BARINEL is the best performing approach. Only for $C = 1$ the BAYES-A approach has equal performance to BARINEL, as for this trivial case the approximations for the h_j are exact. For $C \geq 2$ the plots confirm that BARINEL has superior performance, demonstrating that an exact estimation of h_j is quite relevant. The more challenging the diagnostic problem becomes (higher fault densities), the more BARINEL stands out compared to the statistical approaches and the previous Bayesian reasoning approaches.

V. EMPIRICAL EVALUATION

In this section, we evaluate the diagnostic capabilities and efficiency of the diagnosis techniques for real programs.

A. Experimental Setup

For evaluating the performance of our approach we use the well-known Siemens benchmark set, as well as `gzip`, `sed` and `space` (obtained from SIR [10]). The Siemens suite is composed of seven programs. Every single program has a correct version and a set of faulty versions of the same program. Although the faulty may span through multiple statements and/or functions, each faulty version contains exactly one fault. For each program a set of inputs is also provided, which were created with the intention to test full coverage. In particular, the `Space` package provides $1,000$ test suites that consist of a random selection of (on average) 150 test cases out of $13,585$ and guarantees that each branch of the program is exercised by at least 30 test cases. In our experiments, the test suite used is randomly chosen from the $1,000$ suites provided. Table II provides more information about the programs used in your experiments, where M corresponds to the number of lines of code (components in this context).

For our experiments, we have extended the subject programs with program versions where we can activate arbitrary combinations of *multiple* faults. For this purpose, we limit ourselves to a selection of 143 out of the 183 faults, based on criteria such as faults being attributable to a single line of code, to enable unambiguous evaluation.

Program	Faulty Versions	M	N	Description
`print_tokens`	7	539	4,130	Lexical Analyzer
`print_tokens2`	10	489	4,115	Lexical Analyzer
`replace`	32	507	5,542	Pattern Recognition
`schedule`	9	397	2,650	Priority Scheduler
`schedule2`	10	299	2,710	Priority Scheduler
`tcas`	41	174	1,608	Altitude Separation
`tot_info`	23	398	1,052	Information Measure
`space`	38	9,564	150	ADL Interpreter
`gzip-1.3`	7	5,680	210	Data compression
`sed-4.1.5`	6	14,427	370	Textual manipulator

Table II
THE SUBJECT PROGRAMS

As each program suite includes a correct version, we use the output of the correct version as reference. We characterize a run as failed if its output differs from the corresponding output of the correct version, and as passed otherwise.

B. Performance Results

In this section we evaluate the diagnostic capabilities of BARINEL and compare it with several fault localization techniques. We first evaluate the performance in the context of single faults, and then for multiple fault programs.

1) Single Faults: We compare BARINEL with several well-known statistics-based techniques which have used the Siemens benchmark set described in the previous section. Although the set comprises 132 faulty programs, two of these programs, namely version 9 of `schedule2` and version 32 of `replace`, are discarded as no failures are observed. Besides, we also discard versions 4 and 6 of `print_tokens` because the faults are not in the program itself but in a header file. In summary, we discarded 4 versions out of 132 provided by the suite, using 128 versions in our experiments. For compatibility with previous work in (single-) fault localization, we use the effort/*s*core metric [4], [29] which is the percentage of statements that need to be inspected to find the fault - in other words, the rank position of the faulty statement divided by the total number of statements. Note that some techniques such as in [24], [29] do not rank all statements in the code, and their rankings are therefore based on the program dependence graph of the program.

Figure 3 plots the percentage of located faults in terms of debugging effort. Apart from Ochiai and Tarantula, the following techniques are also plotted: Intersection and Union [29], Delta Debugging (DD) [37], Nearest Neighbor (NN) [29], Sober [24], PPDG [6], and CrossTab [34], which are amongst the best statistics-based techniques (see Section VI). In the single fault context, as mentioned in the previous section, BAYES-A performs equally well as BARINEL. As Sober is publicly available, we run it in our own environment. The values for the other techniques are, however, directly taken from their respective papers. From

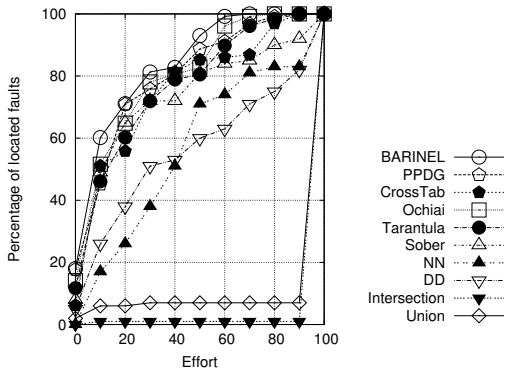


Figure 3. Effectiveness Comparison ($C = 1$)

Figure 3, we conclude that BARINEL is consistently the best performing technique, finding 60% of the faults by examining less than 10% of the source code. For the same effort, using Ochiai would lead a developer to find 52% of the faulty versions, and with Tarantula only 46% would be found. For an effort of less than 1% PPDG performs equally well as BARINEL. Our approach outperforms Ochiai, which is consistently better than Sober and Tarantula. The former two yield similar performance, as also concluded in [24]. Finally, the other techniques plotted are clearly outperformed by the spectrum-based techniques.

The reason for BARINEL's superiority for single-fault programs is established in terms of the following theorem.

Theorem *For single-fault programs, given the available set of observations (A, e), the diagnostic ranking produced by* BARINEL *is theoretically optimal.*

Proof In the single-fault case, the maximum likelihood estimation for h_j reduces from a numerical procedure to a simple analytic expression given by

$$h_j = \frac{n_{10}(j)}{n_{10}(j) + n_{11}(j)} = \frac{x(j)}{x(j) + 1}$$

as, by definition, h_j is the pass fraction of the runs where c_j is involved. Consequently, $\Pr(d_k|e)$ can be written as

$$\Pr(d_k|e) = h_j^{x(j) \cdot n_{11}(j)} \cdot (1 - h_j)^{n_{11}(j)}$$

where $x(j) = n_{10}(j)/n_{11}(j)$. For $C = 1$ where a run fails the faulty component *has* to be involved. In BARINEL, when a candidate does not explain all failing runs its probability is set to 0 as a result from the consistency-based reasoning within BARINEL (cf. the 0-clause of Eq. (4)). This implies that for the remaining candidates $n_{11}(j)$ equals the number of failing runs, which is independent of j. Hence, with respect to the *ranking* the constant $n_{11}(j)$ can be ignored, yielding

$$\Pr'(d_k|e) = \frac{x(j)^{x(j)}}{(x(j) + 1)^{(x(j)+1)}}$$

Since $x(j) > 0$, $\Pr'(d_k|e)$, and therefore $\Pr(d_k|e)$, is monotonically decreasing with $x(j)$ and therefore with h_j. Consequently, the ranking in D equals the (inverse) ranking of h_j. As the maximum likelihood estimator for h_j is perfect by definition, the ranking returned by BARINEL is optimal. □

While the above theorem establishes BARINEL's optimality, the following corollary describes the consequences with respect to similarity coefficients.

Corollary *For single-fault programs, any similarity coefficient that includes $n_{10}(j)$ in the denominator is optimal, provided components c_j are removed from the ranking for which $n_{01}(j) \neq 0$.*

Proof From the above theorem it follows that the ranking in terms of h_j is optimal for the subset of components indicted by the reasoning process, i.e., those components that are always involved in a failing run. The latter condition implies that only components for which $n_{01}(j) = 0$ can be considered. The former implies that (for this subset) the similarity coefficient

$$s(j) = 1 - h_j = \frac{n_{11}(j)}{n_{11}(j) + n_{10}(j)}$$

is optimal. As for the components subset $n_{11}(j)$ is constant, $n_{10}(j)$ determines the ranking while $n_{01}(j)$ plays no role. As all similarity coefficients have an $n_{11}(j)$ term in the numerator, it follows that as long as $n_{10}(j)$ is present in the denominator (the only term that varies with j), such a coefficient yields the same, optimal, ranking as the above BARINEL expression for $s(j)$. □

Experiments using the $n_{01}(j) = 0$ "reasoning" filter, combined with a simple similarity coefficient such as Tarantula or Ochiai indeed confirm that this approach leads to the best performance [32] (equal to BARINEL).

2) Multiple Faults: We now proceed to evaluate our approach in the context of multiple faults, using our extended Siemens benchmark set, `gzip`, `sed`, and `space`. In contrast to Section V-B1 we only compare with the same techniques as in Section IV (BAYES-A, Tarantula, and Ochiai) as for the other related work no data for multiple-fault programs are available (also see Section VI). Similar to Section IV, we aimed at $C = 5$ for the multiple fault-cases, but for `print_tokens` insufficient faults are available. All measurements except for the four-fault version of `print_tokens` are averages over 100 versions, or over the maximum number of combinations available, where we verified that all faults are active in at least one failed run.

Table III presents a summary of the diagnostic quality of the different techniques. The diagnostic quality is quantified in terms of wasted debugging effort W (see Section IV for an explanation of the difference between wasted effort and effort). Again, the results confirm that on average BARINEL

outperforms the other approaches, especially considering the fact that the variance of W is considerably higher (coefficient of variance up to 0.5 for `schedule2`) than in the synthetic case (1,000 sample matrices versus up to 100 matrices in the experiments with real software programs). Only in 3 out of 30 cases, BARINEL is not on top. Apart from the obvious sampling noise (variance), this is due to particular properties of the programs. Using the paired two-tailed Student's t-test, we verified that the differences in the means of W are not significant for those cases in which BARINEL does not clearly outperforms the other approaches, and thus noise is the cause for the small differences in terms of W. As an example, for `print_tokens2` with $C = 2$ the differences in the means are significant, but it is not the case for `schedule` with $C = 1$. For `tcas` with $C = 2$ and $C = 5$, BAYES-A marginally outperforms BARINEL (by less than 0.5%), Ochiai being the best performing approach. This is caused by the fact that (1) the program is almost branch-free and small ($M = 174$) combined with large sampling noise (σ_W = 5% for `tcas`), and (2) almost all failing runs involve all faulty components (highly correlated occurrence). Hence, the program effectively has a single fault spreading over multiple lines. In contrast to the results in the previous section, the performance of BAYES-A and BARINEL for $C = 1$ differ because we also consider valid multiple-fault diagnosis candidates (in the previous section, we only ranked single-fault diagnosis candidates).

Our results show that W decreases with increasing program size (M). This confirms our expectation that the effectiveness of automated diagnosis techniques generally improves with program size. As an illustration, near-zero wasted effort is measured in experiments with SFL on a 0.5 MLOC industrial software product, reported in [40], where the problem reports (tests) typically focus on a particular anomaly (small C).

C. Time/Space Complexity

In this section we report on the time/space complexity of BARINEL, compared to other fault localization techniques. We measure the time efficiency by conducting our experiments on a 2.3 GHz Intel Pentium-6 PC with 4 GB of memory. As most fault localization techniques have been evaluated in the context of single faults, in order to allow us to compare our fault localization approach to related work we limit ourselves to the original, single-fault Siemens benchmark set, which is the common benchmark set to most fault localization approaches. We obtained timings for PPDG and DD from published results [6], [37].

Table IV summarizes the results of the study. The columns show the programs, the average CPU time (in seconds) of BARINEL, BAYES-A, Tarantula/Ochiai, PPDG, and DD, respectively. As expected, the less expensive techniques are the statistics-based techniques Tarantula and Ochiai. At the other extreme are PPDG and DD. BARINEL costs less than PPDG and DD. For example, BARINEL requires less than 10 seconds on average for `replace`, whereas PPDG needs 6 minutes and DD needs approximately 1 hour to produce the diagnostic report. Note that our implementation of BARINEL has not been optimized (the gradient ascent algorithm). This explains the fact that BARINEL is more expensive than the other approximated Bayesian approach. The effect of the gradient ascent costs is clearly noticeable for the first three programs, and is due to a somewhat lower convergence speed as a result of the fact that the h_j are close to 1. Note, that by using a procedure with quadratic convergence this difference would largely disappear (e.g., 100 iterations instead of 10,000, gaining two orders of magnitude). Therefore, the efficiency results should not be viewed as definitive. Experiments using the extended Siemens benchmark set to accommodate multiple faults also show the same trend.

Program	BARINEL	BAYES-A	Tarantula/Ochiai	PPDG	DD
`print_tokens`	24.3	4.2	0.37	846.7	2590.1
`print_tokens2`	19.7	4.7	0.38	243.7	6556.5
`replace`	9.6	6.2	0.51	335.4	3588.9
`schedule`	4.1	2.5	0.24	77.3	1909.3
`schedule2`	2.9	2.5	0.25	199.5	7741.2
`tcas`	1.5	1.4	0.09	1.7	184.8
`tot_info`	1.5	1.2	0.08	97.7	521.4
`space`	41.4	7.4	0.15	N/A	N/A
`gzip`	28.1	6.2	0.19	N/A	N/A
`sed`	92.0	9.7	0.36	N/A	N/A

Table IV
DIAGNOSIS COST FOR THE SINGLE-FAULT SUBJECT PROGRAMS (TIME IN SECONDS)

In the following we interpret the above cost measurements from a complexity point of view. The statistical techniques (such as Tarantula and Ochiai) update the similarity computation (a few scalar operations) per component and per row of the matrix ($O(N \cdot M)$). Subsequently, the report is ordered ($O(M \cdot \log M)$). Consequently, the time complexity is $O(N \cdot M + M \cdot \log M)$. In contrast to the M components in statistical approaches, the Bayesian techniques update $|D|$ candidate probabilities where $|D|$ is determined by STACCATO. Although in all our measurements a constant $|D| = 100$ suffices [2], it is not unrealistic to assume that for very large systems $|D|$ would scale with M, again, yielding $O(N \cdot M)$ for the probability updates. However, there are two differences with the statistical techniques, (1) the cost of STACCATO and (2) in case of BARINEL, the cost of the maximization procedure. The complexity of STACCATO is estimated to be $O(N \cdot M)$ (for a constant matrix density r) [2]. The complexity of the maximization procedure appears to be rather independent of the size of the expression (i.e., M and C) reducing this term to a constant. As, again, the report is ordered, the time complexity again equals $O(N \cdot M + M \cdot \log M)$, putting the Bayesian approaches in the same complexity class as the statistical approaches

		print_tokens			print_tokens2			replace			schedule			schedule2		
	C	1	2	4	1	2	5	1	2	5	1	2	5	1	2	5
	versions	4	6	1	10	43	100	23	100	100	7	20	11	9	35	91
MBD	BAYES-A	1.2	2.4	4.8	5.1	8.9	15.5	3.0	5.2	12.4	0.8	1.5	3.1	21.5	29.4	35.6
MBD	BARINEL	**1.2**	**2.4**	**4.4**	**1.9**	**3.4**	**6.6**	**3.0**	**5.0**	**11.9**	**0.8**	**1.5**	**3.0**	**21.5**	**28.1**	**34.9**
SFL	Ochiai	2.6	5.3	11.5	3.9	7.0	13.5	3.0	5.6	12.4	1.1	2.0	3.7	21.5	29.1	35.5
SFL	Tarantula	7.3	13.2	21.0	6.0	10.4	17.8	4.5	7.7	14.9	1.5	2.7	5.4	23.5	31.4	38.3

		tcas			tot_info			space			gzip			sed		
	C	1	2	5	1	2	5	1	2	5	1	2	5	1	2	5
	versions	30	100	100	19	100	100	28	100	100	7	21	21	5	10	1
MBD	BAYES-A	16.7	24.1	30.5	6.1	11.7	20.9	2.2	3.7	9.9	1.3	2.7	6.7	0.7	0.6	1.4
MBD	BARINEL	16.7	24.5	30.7	**5.0**	**8.5**	**15.8**	**1.7**	**3.0**	**7.4**	**1.0**	**1.9**	**4.3**	**0.3**	**0.4**	**1.4**
SFL	Ochiai	**15.5**	**22.0**	**27.4**	5.2	9.1	16.5	1.7	3.6	8.6	1.3	2.7	7.4	0.4	0.7	1.7
SFL	Tarantula	16.1	22.8	31.6	6.9	11.4	19.4	3.4	6.5	13.9	2.6	5.0	11.4	0.4	0.8	1.7

Table III
WASTED EFFORT W [%] ON COMBINATIONS OF $C = 1 - 5$ FAULTS FOR THE SUBJECT PROGRAMS

modulo a large factor.

With respect to space complexity, statistical techniques need two store the counters $(n_{11}, n_{10}, n_{01}, n_{00})$ for the similarity computation for all M components. Hence, the space complexity is $O(M)$. BAYES-A also stores similar counters but per diagnosis candidate. Assuming that $|D|$ scales with M, these approaches have $O(M)$ space complexity. BARINEL is slightly more expensive because for a given diagnosis d_k it stores the number of times a combination of faulty components in d_k is observed in passed runs $(2^{|d_k|} - 1)$ and in failed runs $(2^{|d_k|} - 1)$. Thus, BARINEL's space complexity is estimated to be $O(2^C \cdot M)$ - being slightly more complex than SFL. In practice, however, memory consumption is reasonable (e.g., around 5.3 MB for `sed`, the largest program used in our experiments).

VI. RELATED WORK

In model-based (logic) reasoning approaches to automatic debugging the model of the program under analysis is typically generated using static analysis. In the work of Mayer and Stumptner [26] an overview of techniques to automatically generate program models from the source code is given, concluding that models generated by means of abstract interpretation [25] are the most accurate for debugging. Model-based approaches include the Δ-slicing and `explain` work of Groce [13], the work of Wotawa, Stumptner, and Mayer [35], and the work of Yilmaz and Williams [36]. Although model-based diagnosis inherently considers multiple faults, thus far the above software debugging approaches only consider single faults. Apart from this, our approach differs in the fact that we use program spectra as dynamic information on component activity, which allows us to exploit execution behavior, unlike static approaches. Besides, our approach does not rely on the approximations required by static techniques (i.e., incompleteness). Most importantly, our approach is less complex, as can also be deduced by the limited set of programs used by the model-based techniques (as an indication, from the Siemens set, these techniques can only handle `tcas` which is the smallest program).

As mentioned earlier, statistical approaches are very attractive from complexity-point of view. Well-known examples are the Tarantula tool by Jones, Harrold, and Stasko [20], the Nearest Neighbor technique by Renieris and Reiss [29], the Sober tool by Lui, Yan, Fei, Han, and Midkiff [24], PPDG by Baah, Podgurski, and Harrold [6], CrossTab by Wong, Wei, Qi, and Zap [34], the Cooperative Bug Isolation by Liblit and his colleagues [22], [39], the Ochiai coefficient by Abreu, Zoeteweij, and Van Gemund [4], the work of Stantelices, Jones, Yu, and Harrold [30], and the work of Wang, Cheung, Chan, and Zhang [33]. Although differing in the way they derive the statistical fault ranking, all techniques are based on measuring program spectra. Examples of other techniques that do not require additional knowledge of the program under analysis are the dynamic program slicing technique by Zhang, He, Gupta, and Gupta [38] and the state-altering approaches Delta Debugging technique by Zeller [37], Hierarchical Delta Debugging approach (HDD) by Misherghi and Zhendong Su [27], and Value-based replacement from Jeffrey, Gupta, and Gupta [17]. The DEPUTO framework by Abreu, Mayer, Stumptner, and Van Gemund [1] combines SFL with a MBD approach [25], where the latter is used to refine the SFL's ranking obtained filtering out candidates that do not explain observed failures when the programs semantics is considered. BARINEL solves the complexity problem in MBD, by taking a spectrum-based approach to MBD, thus scaling to large programs.

Essentially all of the above work have mainly been studied in the context of single faults, except for recent work by Liu, Yan, Fei, and Midkiff [23], Jones, Bowring, and Harrold [19], Abreu, Zoeteweij, and Van Gemund [3], and Steimann and Bertchler [31], who all take an explicit multiple-fault, spectrum-based approach. The work in [23]

proposes two pairwise distance metrics for clustering (failed) test cases that refer to the same fault, after which Sober is used to each cluster of test cases. Unlike BARINEL, this work is not fully automatic, requiring the developer to interpret results during the clustering process. The work in [19] employs clustering techniques to identify traces (rows in A) which refer to the same fault, after which Tarantula is applied to each cluster of rows. Unlike in [23], the approach in [19] is fully automated and does not need the results to be interpreted by the developer. In these clustering approaches there is a possibility that multiple developers will still be effectively fixing the same bug. Our work differs from the above in that we do not seek to engage multiple developers in finding bugs (sequential/iterative approach as opposed to parallel), but in enriching the ranking with multiple-fault diagnosis candidates information that allows one developer to find all bugs quickly. At present, we are unable to empirically compare our approach with [19] as (1) no implementation of the approach is available for experimental comparison [18], and (2) results are only published on `space`, which, in addition, are not reported using established effort metrics (unlike, e.g., PPDG and Delta Debugging).

The significant difference between our previous work in [3] and our approach in this paper is (1) the maximum likelihood health estimation algorithm, replacing the previous, approximate approach, and (2) the use of the STACCATO heuristic reasoning algorithm to bound the number of multiple-fault candidates. In [31] another ranking mechanism is introduced for diagnosis candidates that are derived using a similar technique as in [3], which has exponential time complexity. Therefore, it does not scale well to the set of programs used in our experimental setup.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a multiple-fault localization technique, coined BARINEL, which is based on the dynamic, spectrum-based approach from statistical fault localization methods, combined with a probabilistic reasoning approach from model-based diagnosis. BARINEL employs low-cost, approximate reasoning, employing a novel, maximum likelihood estimation approach to compute the health probabilities per component, at a time and space complexity that is comparable to current SFL approaches due to the use of a heuristic MHS algorithm (STACCATO) underlying the candidate generation process. As a result, BARINEL can be applied to large programs without problems, in contrast to other MBD approaches.

Next to the formal proof of BARINEL's optimality in the single-fault case, synthetic experiments with multiple injected faults have confirmed that our approach consistently outperforms statistical spectrum-based approaches, and our previous Bayesian reasoning approach. Application to a set of software programs also indicates BARINEL's advantage (27 wins out of 30 trials, despite the significant variance), while the exceptions can be pointed to particular program properties in combination with sampling noise.

Future work includes extending the activity matrix from binary to integer, to exploit component involvement frequency (e.g., program loops), reducing the cost of gradient ascent by introducing quadratic convergence techniques, and studying the influence of different program spectra on the diagnostic quality of BARINEL.

ACKNOWLEDGMENT

We extend our gratitude to Rafi Vayani for his experiments showing the optimality of the reasoning approach in the single-fault case. Furthermore, we gratefully acknowledge the collaboration with our TRADER project partners.

REFERENCES

[1] R. Abreu, W. Mayer, M. Stumptner, and A. J. C. van Gemund. Refining spectrum-based fault localization rankings. In *Proceedings of the Annual Symposium on Applied Computing (SAC'09)*.

[2] R. Abreu and A. J. C. van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *Proceedings of Symposium on Abstraction, Reformulation, and Approximation (SARA'09)*.

[3] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. An observation-based model for fault localization. In *Proceedings of Workshop on Dynamic Analysis (WODA'08)*.

[4] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings of The Testing: Academic and Industrial Conference - Practice and Research Techniques (TAIC PART'07)*.

[5] M. Avriel. *Nonlinear Programming: Analysis and Methods*. 2003.

[6] G. K. Baah, A. Podgurski, and M. J. Harrold. The probabilistic program dependence graph and its application to fault diagnosis. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'08)*.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2nd edition.

[8] J. de Kleer. Diagnosing intermittent faults. In *Proceedings of International Workshop on Principles of Diagnosis (DX'07)*, May.

[9] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

[10] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering: An International Journal*, 10(4):405–435, 2005.

[11] A. Feldman, G. Provan, and A. J. C. van Gemund. Computing minimal diagnoses by greedy stochastic search. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI'08)*.

[12] A. Feldman and A. J. C. van Gemund. A two-step hierarchical algorithm for model-based diagnosis. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI'06)*.

[13] A. Groce. Error explanation with distance metrics. In *Proceedings of International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*.

[14] N. Gupta, H. He, X. Zhang, and R. Gupta. Locating faulty code using failure-inducing chops. In *Proceedings of International Conference on Automated Software Engineering (ASE'05)*.

[15] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation of program spectra. In *Proceedings of International Workshop on Program Analysis for Software Tools and Engineering (PASTE'98)*.

[16] T. Janssen, R. Abreu, and A. J. C. van Gemund. ZOLTAR: A toolset for automatic fault localization. In *Proceedings of the International Conference on Automated Software Engineering (ASE'09) - Tool Demonstrations*.

[17] D. Jeffrey, N. Gupta, and R. Gupta. Fault localization using value replacement. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'08)*.

[18] J. A. Jones. Personal communication, April 2009.

[19] J. A. Jones, J. F. Bowring, and M. J. Harrold. Debugging in parallel. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'07)*.

[20] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of International Conference on Software Engineering (ICSE'02)*.

[21] C. Lattner and V. S. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO'04)*.

[22] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proceedings of Conference on Programming Language Design and Implementation (PLDI'05)*.

[23] C. Liu and J. Han. Failure proximity: A fault localization-based approach. In *Proceedings of Symposium on the Foundations of Software Engineering (FSE'06)*.

[24] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff. Sober: Statistical model-based bug localization. In *Proceedings of European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE-13)*.

[25] W. Mayer and M. Stumptner. Abstract interpretation of programs for model-based debugging. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'07)*.

[26] W. Mayer and M. Stumptner. Evaluating models for model-based debugging. In *Proceedings of International Conference on Automated Software Engineering (ASE'08)*.

[27] G. Misherghi and Z. Su. HDD: Hierarchical delta debugging. In *Proceedings of International Conference on Software Engineering (ICSE'06)*.

[28] J. Pietersma and A. J. C. van Gemund. Temporal versus spatial observability tradeoffs in model-based diagnosis. In *Proceedings of International Conference on Systems, Man, and Cybernetics (SMC'06)*.

[29] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Proceedings of International Conference on Automated Software Engineering (ASE'03)*.

[30] R. Santelices, J. A. Jones, Y. Yu, and M. J. Harrold. Lightweight fault-localization using multiple coverage types. In *Proceedings of International Conference on Software Engineering (ICSE'09)*.

[31] F. Steimann and M. Bertchler. A simple coverage-based locator for multiple faults. In *Proceedings of International Conference on Software Testing and Verification (ICST'09)*.

[32] R. Vayani. Improving automatic software fault localization, Delft University of Technology, July 2007. Master's thesis.

[33] X. Wang, S. Cheung, W. Chan, and Z. Zhang. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In *Proceedings of International Conference on Software Engineering (ICSE'09)*.

[34] W. Wong, T. Wei, Y. Qi, and L. Zhao. A crosstab-based statistical method for effective fault localization. In *Proceedings of International Conference on Software Testing and Verification (ICST'08)*.

[35] F. Wotawa, M. Stumptner, and W. Mayer. Model-based debugging or how to diagnose programs automatically. In *Proceedings of International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE'02)*.

[36] C. Yilmaz and C. Williams. An automated model-based debugging approach. In *Proceedings of International Conference on Automated Software Engineering (ASE'07)*.

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

[38] X. Zhang, H. He, N. Gupta, and R. Gupta. Experimental evaluation of using dynamic slices for fault location. In *Proceedings of International Workshop on Automated and Analysis-Driven Debugging (AADEBUG'05)*.

[39] A. X. Zheng, M. I. Jordan, B. Liblit, M. Naik, and A. Aiken. Statistical debugging: Simultaneous identification of multiple bugs. In *Proceedings of International Conference on Machine Learning (ICML'06)*.

[40] P. Zoeteweij, R. Abreu, R. Golsteijn, and A. J. C. van Gemund. Diagnosis of embedded software using program spectra. In *Proceedings of International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'07)*.