Artificial Intelligence 174 (2010) 1481–1497

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Diagnosing multiple intermittent failures using maximum likelihood estimation ☆

Rui Abreu [a,*], Arjan J.C. van Gemund [b]

[a] Department of Informatics Engineering, Faculty of Engineering, University of Porto, Portugal
[b] Embedded Software Group, Delft University of Technology, Faculty of Electrical Eng., Math., and CS, The Netherlands

ARTICLE INFO

Article history:
Received 14 September 2009
Received in revised form 8 September 2010
Accepted 8 September 2010
Available online 25 September 2010

Keywords:
Fault diagnosis
Bayesian reasoning
Maximum likelihood estimation

ABSTRACT

In fault diagnosis intermittent failure models are an important tool to adequately deal with realistic failure behavior. Current model-based diagnosis approaches account for the fact that a component c_j may fail intermittently by introducing a parameter g_j that expresses the probability the component exhibits correct behavior. This component parameter g_j, in conjunction with a priori fault probability, is used in a Bayesian framework to compute the posterior fault candidate probabilities. Usually, information on g_j is not known a priori. While proper estimation of g_j can be critical to diagnostic accuracy, at present, only approximations have been proposed. We present a novel framework, coined BARINEL, that computes estimations of the g_j as integral part of the posterior candidate probability computation using a maximum likelihood estimation approach. BARINEL's diagnostic performance is evaluated for both synthetic systems, the Siemens software diagnosis benchmark, as well as for real-world programs. Our results show that our approach is superior to reasoning approaches based on classical persistent failure models, as well as previously proposed intermittent failure models.

1. Introduction

In model-based fault diagnosis (MBD) faults are typically assumed to be persistent. In many practical situations, however, faults exhibit intermittent failure behavior, such as in copiers where sometimes sheets may be blank, or where a worn roller sometimes slips and causes a paper jam [12]. Intermittent failure is also relevant in software fault diagnosis, which is the primary context of this paper. Although software is supposed to be inherently deterministic, intermittent failure models are often essential. This can be due to non-determinism (e.g., race conditions) caused by design faults related to properly dealing with concurrency. A more compelling reason is the modeling abstraction applied when reasoning about software components. In our reasoning approach the input and output values are abstracted to binary "correctness" values. Consider, the integer division `x/10` where `10` is a fault that should have been `15`. Consider two input values `x = 15`, and `x = 20`, respectively. In the first case, the component produces a correct output, whereas in the second case the component fails. If both inputs were modeled as correct (e.g., because they were produced by other, nominal, components) the division component exhibits intermittent failure behavior. Although a weak fault model (that does not stipulate particular fault behavior) admits any output behavior, modeling inconsistently failing (software) components merely in terms of weak models results in degraded diagnostic performance [4].

☆ 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.

* Corresponding author.
E-mail addresses: rui@computer.org (R. Abreu), a.j.c.vangemung@tudelft.nl (A.J.C. van Gemund).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2010.09.003

A model for intermittent failure behavior [9] was introduced as an extension of the GDE framework [11,13]. Essentially, next to the prior probability p_j that a component c_j is at fault, a parameter g_j is used to express the probability that a faulty component exhibits correct (good, hence g) behavior ($g_j = 0 =$ for persistently failing, $g_j = 1 =$ effectively ok, $0 < g_j < 1 =$ intermittently failing). The model is incorporated into the standard, Bayesian framework that computes the posterior probability of diagnosis candidates based on observations [8,13].

The intermittent failure model has been shown to yield significantly better results (e.g., in the diagnosis and replanning of paper sheet paths in copiers with intermittent component failures [23], and in software fault diagnosis [4]), compared to an approach based on a classical, persistent failure model. An important problem in using the intermittent failure model, however, is the estimation of g_j, as calibration data on correct and incorrect component behavior is typically not available. Estimating g_j for each component c_j would be straightforward when (sufficient) system observations are available where only that *single*, intermittently failing component is involved [9]. However, in a multiple-fault context usually only system observations are available in which *multiple* faulty components are involved. Consequently, isolating to what extent each individual component contributes to the observed, intermittent failure behavior is not trivial. However, as the influence of g_j in the computation of the posterior probability of each diagnostic candidate is significant, exact knowledge of each g_j *can* be critical to overall diagnostic accuracy.

In [12] as well as in [4,5] strategies have been proposed to estimate the g_j in a multiple-fault context. However, the approaches are essentially based on an approximation. In this paper, we present a novel approach to compute the g_j, in conjunction with a new approach towards the computation of the posterior candidate probabilities using an intermittent failure model that generalizes over classical, persistent MBD approaches. The approach represents a departure from the current Bayesian framework as used in current diagnosis approaches (e.g., [4] and [12]) in the sense that (1) the resulting g_j are *maximum likelihood estimators* instead of approximations, and (2) the computation of the posterior candidate probabilities is an *integral product* of the g_j estimation procedure.

Apart from diagnosis accuracy, in this paper we also address diagnosis efficiency. The weak (intermittent) modeling approach, in combination with the large systems we consider (in the order of tens of thousands of components) leads to a huge diagnostic candidate space. In this paper we present a minimal hitting set algorithm that features a novel, diagnosis-specific heuristic that directs the search to generate candidates in order of decreasing posterior probability, even within equal-cardinality groups. This feature allows the candidate generation process to be truncated to a very limited number of candidates (merely 100 in our experiments), yet effectively capturing all posterior probability mass. This tailored algorithm enables us to apply our diagnosis technique to very large systems.

This 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 g_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 present a new algorithm to compute the minimal hitting set from a set of conflicts, called STACCATO,[2] and derive its time and space complexity;
- We compare the accuracy and complexity of BARINEL (including STACCATO) to the current approaches in [4] and [12] for synthetically generated observation series based on injected faults with known g_j setpoints;
- We describe the application of our approach to software multiple-fault diagnosis and evaluate its diagnostic performance using the well-known Siemens suite of benchmark programs (extended for multiple faults) as well as real-world programs (`space`, `sed`, `gzip`).

The results from the synthetic experiments, as well as from the application to real software systems, confirm that our new approach has superior diagnostic performance to all Bayesian approaches to intermittently failing systems known to date, at very limited computation cost.

The paper is organized as follows. In the next section we describe the current Bayesian approach to persistent and intermittent failure models. In Sections 3 and 4 we describe our new approach to candidate generation, and posterior probability computation, respectively. Sections 5 and 6 present experimental results for synthetic observations, and real program codes, respectively. Section 7 describes related work, while Section 8 concludes the paper.

2. Preliminaries

In this section we describe the state-of-the-art in MBD involving intermittent failures.

[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.

[2] STACCATO is an acronym for STAtistiCs-direCted minimAl hiTing set algOrithm.

2.1. Basic definitions

Definition. A diagnostic system DS is defined as the triple $DS = \langle SD, COMPS, OBS\rangle$, where SD is a propositional theory describing the behavior of the system, $COMPS = \{c_1, \ldots, c_M\}$ is a set of components in SD, and OBS is a set of observable variables in SD.

With each component $c_j \in COMPS$ we associate a *health variable* h_j which denotes component health. The health states of a component are either healthy (h_j *true*) or faulty (h_j *false*).

Definition. An h-literal is h_j or $\neg h_j$ for $c_j \in COMPS$.

Definition. An h-clause is a disjunction of h-literals containing no complementary pair of h-literals.

Definition. A conflict of $(SD, COMPS, OBS)$ is an h-clause of negative h-literals entailed by $SD \cup OBS$.

Definition. Let S_N and S_P be two disjoint sets of components indices, faulty and healthy, respectively, such that $COMPS = \{c_j \mid j \in S_N \cup S_P\}$ and $S_N \cap S_P = \emptyset$. We define $d(S_N, S_P)$ to be the conjunction $(\bigwedge_{j\in S_N} \neg h_j) \wedge (\bigwedge_{j\in S_P} h_j)$.

A diagnosis candidate is a sentence describing one possible state of the system, where this state is an assignment of the status healthy or not healthy to each system component.

Definition. A diagnosis candidate $d(S_N, S_P)$ for DS given an observation obs over variables in OBS is such that

$$SD \wedge obs \wedge d(S_N, S_P) \not\models \bot$$

In the remainder we refer to $d(S_N, S_P)$ simply as d, which we identify with the set S_N of indices of the negative literals.

A *minimal diagnosis* is a diagnosis that is not subsumed by another of lower fault cardinality (i.e., number of negative h-literals $C = |d|$). A minimal diagnosis is a minimal hitting set over all conflicts.

Definition. A minimal hitting set of a collection S is a set d such that

$$\forall s_i \in S, \quad s_i \cap d \neq \emptyset \ \wedge \ \nexists d' \subset d\colon s_i \cap d' \neq \emptyset$$

i.e., d contains at least one element from each subset in set S, and no proper subset of d is a hitting set. There may be several minimal hitting sets for S, which constitutes a collection of minimal hitting sets.

Definition. A diagnostic report $D = \langle d_1, \ldots, d_k, \ldots, d_K\rangle$ is an ordered set of all K minimal diagnosis candidates, for which $SD \wedge obs \wedge d_k \not\models \bot$.

The *diagnostic accuracy* of a diagnostic report D depends on the ranking of the actual system's fault state d^*. Assuming a diagnostician traverses D top to bottom, a diagnostic approach that produces a D where d^* is ranked on top has higher accuracy (i.e., generates less testing effort) than an approach that ranks d^* lower. Details are discussed in the experimental evaluation sections later on.

The Bayesian approach serves as the foundation for the derivation of diagnostic candidates, i.e., (1) deducing whether a candidate diagnosis d_k is consistent with the observations, and (2) the posterior probability $\Pr(d_k)$ of that candidate being the actual diagnosis. With respect to (1), rather than computing $\Pr(d_k)$ for *all* possible candidates, just to find that most of them have $\Pr(d_k) = 0$, search algorithms are typically used instead, such as CDA* [32], Safari [15], or just a minimal hitting set (MHS) algorithm when conflict sets are available (e.g. [13]), but the Bayesian probability framework remains the basis. In this section we will briefly describe the contemporary approach to the derivation of candidates and their posterior probability.

2.2. Candidate generation

Consider a particular process, involving a set of components, that either yields a nominal result or a failure. For instance, in a logic circuit a process is the sub-circuit (cone) activity that results in a particular primary output. In a copier a process is the propagation of a sheet of paper through the system. In software a process is the sequence of software component activity (e.g., statements) that results in a particular return value. The result of a process is either nominal ("pass") or an error ("fail"). As explained earlier, in the sequel we assume weak component fault models ($h \Rightarrow \langle$nominal behavior$\rangle$), compatible with the notion of intermittency which allows a faulty component to (intermittently) exhibit correct behavior.

$$
\begin{array}{c} \\ N \text{ sets} \end{array}
\begin{array}{c} M \text{ components} \\ \left[\begin{array}{cccc} a_{11} & a_{12} & \ldots & a_{1M} \\ a_{21} & a_{22} & \ldots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NM} \end{array}\right] \end{array}
\begin{array}{c} \text{conflict} \\ \left[\begin{array}{c} e_1 \\ e_2 \\ \vdots \\ e_N \end{array}\right] \end{array}
$$

Fig. 1. Input to the diagnostic process.

Definition. Let $S_f = \{c_j \mid c_j \text{ involved in a failing process}\}$, and let $S_p = \{c_j \mid c_j \text{ involved in a passing process}\}$, denote the *fail set* and *pass set*, respectively.

Approaches for fault diagnosis that assume persistent, weak failure models often generate candidates based on fail sets (aka conflict sets), essentially using an MHS algorithm to derive minimal candidates. A well-known example is GDE [13] where fail sets are derived from detecting inconsistencies in the system given certain input and output observations. Recent approaches that allow intermittency also take into account pass sets (consistent behavior). Examples that use pass sets next to fail sets include logic circuits with intermittently failing gates, and the copier and software systems mentioned earlier. In software pass and fail sets originate from dynamically profiling the software components (e.g., statements or modules) during each program run. Formally, each component is associated with a weak model

$$h \implies (ok_{inp} \implies ok_{out})$$

where ok_{inp} and ok_{out} denote the (binary) correctness of the component's input and output. The correctness of each program run is modeled by a chain of above component models, where, by definition, ok_{inp} of the first component in the chain is true, and where ok_{out} of the last component reflects whether the run passes or fails. In the former case, a pass set is recorded, whereas in the latter case a fail set (essentially, a conflict set) is recorded. In software, a (pass or fail) set is commonly referred to a *hit spectrum* [3,20].

Note that the amount and quality of the pass and fail sets has a profound effect on diagnostic quality, possibly even more than the intermittency/posterior candidate probability computation schemes that we are addressing in this paper. However, in this paper we do not address pass and fail set generation, and we assume that a number of pass and fail sets have been collected, either through static, model-based techniques (e.g., logic circuits as mentioned earlier), or by spectrum-based techniques as in software. A fail set indicts components (i.e., increases their posterior fault probability), whereas a pass set exonerates components (i.e., decreases their posterior fault probability). The extent of indictment or exoneration is computed using Bayes' rule.

Definition. Let N denote the number of passing and failing processes. Let N_f and N_p, $N_f + N_p = N$, denote the number of fail and pass sets (spectra), respectively. Let A denote the $N \times M$ *activity matrix* of the system, where a_{ij} denotes whether component j was involved in process i ($a_{ij} = 1$) or not ($a_{ij} = 0$). Let e denote the *error vector*, where e_i signifies whether process i has passed ($e_i = 0$) or failed ($e_i = 1$, i.e., a conflict).

The observations (A, e) are the only input to the diagnosis process (see Fig. 1).

In our BARINEL approach we compute the candidates from the fail sets using our STACCATO MHS algorithm (Section 3).

2.3. Candidate probability computation

Given the multitude of candidates that are typically generated, the candidate ranking induced by posterior probability computation is critical to diagnostic accuracy. Let $\Pr(j) = p_j$ denote the prior probability that a component c_j is at fault. Assuming components fail independently the prior probability of a candidate d_k is given by

$$\Pr(d_k) = \prod_{j \in S_N} \Pr(\{j\}) \cdot \prod_{j \in S_P} \big(1 - \Pr(\{j\})\big)$$

For each observation $obs_i = (A_{i*}, e_i)$ the posterior probabilities are updated according to Bayes rule (naive Bayes classifying)

$$\Pr(d_k | obs_i) = \frac{\Pr(obs_i | d_k)}{\Pr(obs_i)} \cdot \Pr(d_k) \tag{1}$$

The denominator $\Pr(obs_i)$ is a normalizing term that is identical for all d_k and thus needs not be computed directly. $\Pr(obs_i | d_k)$ is defined as

$$\Pr(obs_i | d_k) = \begin{cases} 0 & \text{if } obs_i \wedge d_k \text{ are inconsistent;} \\ 1 & \text{if } obs_i \text{ is unique to } d_k; \\ \varepsilon & \text{otherwise.} \end{cases} \tag{2}$$

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

For the large majority of cases, the ε-clause applies. Many policies exist for ε [8]. Three policies can be distinguished. The first policy, denoted $\varepsilon^{(0)}$ equals the classical MBD policy for persistent, weak failures, and is defined as follows

$$\varepsilon^{(0)} = \begin{cases} \frac{E_P}{E_P+E_F} & \text{if } e_i = 0 \\ \frac{E_F}{E_P+E_F} & \text{if } e_i = 1 \end{cases} \tag{3}$$

where $E_P = 2^M$ and $E_F = (2^{|d_k|} - 1) \cdot 2^{M-|d_k|}$ are the numbers of passed and failed observations that can be explained by diagnosis d_k, respectively. A disadvantage of this classical policy is that pass sets, apart from making single faults more probable than multiple faults, do not help in pinpointing the faults, in particular for weak fault models which do not rule out any candidates (the 2^M term in Eq. (3)). In addition, there is no way to distinguish between diagnoses with the same cardinality, because the terms are merely a function of the cardinality of the diagnosis candidate.

The next two policies consider component intermittent failure behavior by accounting for the fact that components involved in pass sets should to some extent be exonerated. In the following we distinguish between two policies, $\varepsilon^{(1)}$ [9] and $\varepsilon^{(2)}$ [4] which are defined as

$$\varepsilon^{(1)} = \begin{cases} g(d_k) & \text{if } e_i = 0 \\ 1 - g(d_k) & \text{if } e_i = 1 \end{cases}$$

and

$$\varepsilon^{(2)} = \begin{cases} g(d_k)^m & \text{if } e_i = 0 \\ 1 - g(d_k)^m & \text{if } e_i = 1 \end{cases}$$

where $m = \sum_{j \in d_k} [a_{ij} = 1]$ is the number of faulty components according to d_k involved in process i.[3] Note that a term $g(d_k)$ is used rather than the real individual component intermittency parameters g_j. As mentioned earlier, this is due to the fact that obtaining g_j from pass and fail sets where multiple intermittent failures are involved has been far from trivial. Instead, an "effective" intermittency parameter $g(d_k)$ is estimated for the multiple-fault candidate d_k by counting how many times components of d_k are involved in pass and fail sets. In both policies $g(d_k)$ is approximated by

$$g(d_k) = \frac{n_{10}(d_k)}{n_{10}(d_k) + n_{11}(d_k)}$$

where

$$n_{10}(d_k) = \sum_{i=1..N} \left[\left(\bigvee_{j \in d_k} a_{ij} = 1 \right) \wedge e_i = 0 \right]$$

$$n_{11}(d_k) = \sum_{i=1..N} \left[\left(\bigvee_{j \in d_k} a_{ij} = 1 \right) \wedge e_i = 1 \right]$$

Policy $\varepsilon^{(2)}$ is a variant of $\varepsilon^{(1)}$, which approximates the probability $\prod_{j \in d_k} g_j$ that all m components in d_k exhibit good behavior by $g(d_k)^m$ assuming that all components of d_k have equal g values. This takes into account the fact that the failure probability should increase with the *number* of faults involved (i.e., the diagnosis should be more probable to be the true fault explanation).

2.4. Example

To illustrate how current Bayesian approaches work, consider the diagnosis candidates d_k in Table 1 obtained from a 2-faulty `gzip` software program (components 2553 and 2763 are faulty, $M = 5680$ components, $N = 210$ test cases, of which $N_F = 12$ failed). For simplicity, we refrain from reporting the activity matrix, summarizing it in terms of $n_{11}(d_k)$, $n_{10}(d_k)$,

[3] $[\cdot]$ is Iverson's operator ($[\text{true}] = 1$, $[\text{false}] = 0$).

Table 1
Candidates obtained from `gzip`.

d_k	n_{11}	n_{10}	n_{01}	n_{00}	$g(d_k)$	
{1347}	12	189	0	9	0.94	
{2553, 2763}	12	16	0	182	0.57	never involved simultaneously
{2682, 2745}	12	2	0	196	0.14	never involved simultaneously
{2110, 2745}	12	2	0	196	0.14	both involved in 2 passed and 8 failed processes

Table 2
Diagnostic reports.

$\varepsilon^{(0)}$		$\varepsilon^{(1)}$		$\varepsilon^{(2)}$	
⋮	⋮	⋮	⋮	⋮	⋮
{1347}	0.11×10^{-1}	⋮	⋮	⋮	⋮
⋮	⋮	{2110, 2745}	0.10×10^{-3}	{2682, 2745}	0.12×10^{-3}
⋮	0.11×10^{-3}	⋮	⋮	⋮	⋮
{2682, 2745}	0.11×10^{-3}	{2682, 2745}	0.11×10^{-3}	{2110, 2745}	0.73×10^{-5}
⋮	⋮	⋮	⋮	⋮	⋮
{2110, 2745}	0.11×10^{-3}	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	**{2553, 2763}**	0.19×10^{-9}
⋮	⋮	**{2553, 2763}**	0.17×10^{-9}	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	{1347}	0.61×10^{-19}	{1347}	0.69×10^{-19}
{2553, 2763}	0.11×10^{-3}	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮

$n_{01}(d_k)$, $n_{00}(d_k)$ instead.[4] The terms $n_{01}(d_k)$ and $n_{00}(d_k)$ are defined as follows

$$n_{01}(d_k) = \sum_{i=1..N} \left[\left(\bigvee_{j \in d_k} a_{ij} = 0 \right) \wedge e_i = 1 \right]$$

$$n_{00}(d_k) = \sum_{i=1..N} \left[\left(\bigvee_{j \in d_k} a_{ij} = 0 \right) \wedge e_i = 0 \right]$$

A snippet of the diagnostic reports obtained for the different policies is given in Table 2. Common to traditional policies, $\varepsilon^{(0)}$ does not distinguish between candidates with the same cardinality, ranking them in order of diagnosis candidate's cardinality. The diagnostic report yielded by $\varepsilon^{(2)}$ differs from $\varepsilon^{(1)}$ because $\varepsilon^{(2)}$ takes into account the number of (faulty) components involved in a process (the rationale being that the more faulty components are involved, the more likely it is that the run will fail). Essentially, due to the ranking position of the true fault $\varepsilon^{(2)}$ requires the developer to inspect less code than the other policies.

In Section 4 we will show that knowledge of the individual g_j yields far better results than the above $g(d_k)$ estimations.

3. Candidate generation: STACCATO

As mentioned earlier, we derive the diagnosis candidates from the activity matrix A comprising the pass and fail sets. As the fail sets represent conflicts, we apply a minimal hitting set algorithm to compute the diagnosis candidates. Due to the typically large number of hitting sets a search heuristic that *focuses* the search towards solutions that are potentially a minimal hitting set will yield significant efficiency gains. However, many of the computed minimal hitting sets may potentially be of little value (i.e., have very low posterior probability). Therefore, the solutions need to be *ordered* in terms of relevance, possibly aborting the search once a particular number of minimal hitting sets have been found, again boosting efficiency. MHS algorithms typically generate candidates in terms of increasing cardinality, implying that cardinalities of highest posterior probability are generated first. However, changes in the order within the same cardinality class (aka ambiguity group)

[4] For interested readers, the activity matrix can be downloaded from http://www.st.ewi.tudelft.nl/~abreu/aij.

can greatly affect diagnostic accuracy. In this section we present our approximate, statistics-directed minimal hitting set algorithm, coined STACCATO, aimed to increase search efficiency.

Similar to contemporary MHS algorithms [13,10,16], the algorithm combines components, starting with cardinality $C = 1$, until a combination is found that covers all conflicts. In principle, the algorithm executes in depth-first order until all minimal hitting sets are found. Initially, the components c_j are ordered using a heuristic function $\mathcal{H} : j \to \mathcal{R}$. Then during the search the algorithm selects the components in that order.

A generally accepted heuristic [13] is to assume that components that are members of more fail sets than other components, are more likely to be part of a minimal hitting set. The trivial case are those components that are involved in all sets, which constitute minimal hitting sets of cardinality 1. This heuristic is given by

$$\mathcal{H}(j) = \sum_{i=1}^{N} a_{ij} \tag{4}$$

Despite its generally good performance, the above heuristic is not particularly tailored to the diagnostic domain. Given a set of conflicts, the MHS solutions should ideally be ordered in terms of Eq. (1). However, due to the circular dependency, we would first need an MHS algorithm to be able to solve Eq. (1), which would defeat any cost-effective approach. Hence, a low-cost heuristic that still provides a good prediction of Eq. (1) is a critical success factor.

A low-cost, statistics-based technique that is known to be a good predictor for ranking (software) components in order of likelihood to be at fault is *spectrum-based fault localization* (SFL) [3]. SFL takes the same (spectral) input (A, e) and produces a ranking of the components in order of fault likelihood. The component ranking is computed using a *similarity coefficient* that measures the statistical correlation between component involvement and erroneous/nominal system behavior. Many similarity coefficients exist for SFL, the best one currently being the Ochiai coefficient, known from molecular biology and introduced to SFL in [3]. It is defined as follows

$$s(j) = \begin{cases} \frac{n_{11}(\{j\})}{den(j) = \sqrt{(n_{11}(\{j\}) + n_{01}(\{j\})) * (n_{11}(\{j\}) + n_{10}(\{j\}))}}, & \text{if } den(j) \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The similarity coefficient indicts components using $n_{11}(\{j\})$, and exonerates components using $n_{10}(\{j\})$ and $n_{01}(\{j\})$. In [3] it has been shown that similarity coefficients provide an ordering of components that yields good diagnostic accuracy, i.e., components that rank highest are usually faulty. This diagnostic performance, combined with the very low complexity of $s(j)$ is the key motivation to use the Ochiai coefficient $s(j)$ for $\mathcal{H}$. If (A, e) only contains conflicts (i.e., $\nexists e_i = 0$), the ranking returned by this heuristic function reduces to the one produced by the more simple equation (4) and, therefore, classic MHS problems are also adequately handled by our MBD-specific heuristic. STACCATO uses the SFL heuristic equation (5) to focus the search of the minimal hitting set computation (see Algorithm 1).

Algorithm 1 STACCATO

Inputs: Matrix (A, e), number of components M, stop criteria λ, L

Output: Minimal Hitting set D

```
1: T_F ← {A_{i*} | e_i = 1}        ▷ Collection of conflict sets
2: R ← RANK(H, A, e)               ▷ Rank according to heuristic H
3: D ← ∅
4: seen ← 0
5: for all j ∈ {1..M} do
6:    if n_11({j}) = |T_F| then
7:       PUSH(D, {j})
8:       A ← STRIP_COMPONENT(A, j)
9:       R ← R\{j}
10:      seen ← seen + 1/M
11:   end if
12: end for
13: while R ≠ ∅ ∧ seen ⩽ λ ∧ |D| ⩽ L do
14:   j ← POP(R)
15:   seen ← seen + 1/M
16:   (A', e') ← STRIP(A, e, j)
17:   D' ← STACCATO(A', e', M − |{j | n_11(j) = |T_F|}| − 1, λ, L)
18:   while D' ≠ ∅ do
19:      j' ← POP(D')
20:      j' ← {j} ∪ j'
21:      if IS_NOT_SUBSUMED(D, j') then
22:         PUSH(D, j')
23:      end if
24:   end while
25: end while
26: return D
```

STACCATO is an algorithm that is recursively applied to substructures of (A, e) while generating the minimal hitting sets. The algorithm features two phases. The first phase processes components which, at that level of recursion, are minimal hitting sets themselves (lines 5–12). These components are stored until in the second phase of the algorithm the components are (possibly) joined to form a minimal hitting set of (A, e). The second phase processes components which are no hitting set, and are only involved in a subset of the fail sets. For such a component, other components are searched which, together, form a hitting set. This is achieved by recursively calling STACCATO for the substructure of (A, e) from which the component has been removed, together with the fail sets that were already covered by that component (lines 16, and 17, respectively). The MHS found is finally checked for subsumption with respect to the MHS found earlier (line 21). If the new MHS is already subsumed by an earlier MHS in D the new MHS is discarded. Conversely, if earlier MHS in D are subsumed by the new MHS, the earlier MHS are removed from D.

To illustrate how STACCATO works, consider the following (A, e), comprising four fail sets and three pass sets.

c_1	c_2	c_3	c_4	c_5	e_i	
1	0	0	1	1	1	(conflict)
0	1	1	0	1	1	(conflict)
0	0	1	0	1	1	(conflict)
1	0	0	0	1	1	(conflict)
0	0	1	1	1	0	(nominal)
0	0	1	1	1	0	(nominal)
0	0	1	1	1	0	(nominal)

From (A, e) it follows $\mathcal{H}(1) = 0.7$, $\mathcal{H}(2) = 0.5$, $\mathcal{H}(3) = 0.4$, $\mathcal{H}(4) = 0.3$, and $\mathcal{H}(5) = 0.8$, yielding the following ranking $\langle 5, 1, 2, 3, 4\rangle$. As component c_5 is involved in all failed sets, it is added to D and removed from A using the STRIP_COMPONENT function, avoiding solutions subsumed by $\{5\}$ to be considered (lines 5–12). After this phase (A, e) is as follows

c_1	c_2	c_3	c_4	e_i
1	0	0	1	1
0	1	1	0	1
0	0	1	0	1
1	0	0	0	1
0	0	1	1	0
0	0	1	1	0
0	0	1	1	0

Next component to be checked is c_1, which is not involved in two failed sets. Thus, in order to prepare for STACCATO to find other components to form a new hitting set (rationale is to find components that are involved in fail sets in which current component is not involved), the column for that component as well as all fail sets in which it is involved are removed from (A, e), using the STRIP function, yielding the following

c_2	c_3	c_4	e_i
1	1	0	1
0	1	0	1
0	1	1	0
0	1	1	0
0	1	1	0

Running STACCATO with the newly generated (A, e) yields the following ranking $\langle 2, 3\rangle$ (line 17). As c_2 is not involved in all failed sets, and again to prepare for STACCATO to find other components to form a new hitting set, the column for that component as well as all fail sets in which it is involved are removed from (A, e), using the STRIP function, yielding the following

c_3	c_4	e_i
1	0	1
1	1	0
1	1	0
1	1	0

Running STACCATO with the newly generated (A, e) yields a ranking with component 3 only, which is an MHS for the current (A, e). For each MHS d returned by this invocation of STACCATO, the union of d and c_1, c_2 is checked ($\{1, 2\}$), and because

this set is involved in all failed sets, and is not subsumed by any other candidate in D, it is also added to the list of solutions D (lines 18–24). At this point, $D = \{\{5\}, \{1, 2, 3\}\}$.

Once all combinations with c_2 are checked, STACCATO considers the next component in the ranking ($\langle 2, 3\rangle$). As c_3 is involved in all failed sets, STACCATO will check if c_3 together with c_1 ($\{1, 3\}$) either subsumes or is subsumed by any other candidate in D (line 21). As $\{1, 2, 3\}$ is subsumed by $\{1, 3\}$, $\{1, 2, 3\}$ is removed from D and $\{1, 3\}$ is added. After this step, $D = \{\{5\}, \{1, 3\}\}$.

In subsequent iterations of the STACCATO algorithm, the following three sets are also considered: $\{2, 1, 3\}$, $\{3, 1\}$, and $\{4, 1, 3\}$. However, as they are subsumed by $\{1, 3\}$ (line 21) they are discarded. Therefore, STACCATO eventually returns $D = \{\{5\}, \{1, 3\}\}$.

In the above example the two parameters were set at $\lambda = 1$ (relative search depth), and $L = \infty$ (maximum number of solutions returned) which results in a full search (i.e., STACCATO is complete). In practice, however, we exploit STACCATO's focusing property by (1) decreasing λ such that only the top fraction λ of components in the ranking are actually selected in each recursion, and (2) aborting the search after L solutions have been returned. Experiments for synthetic problems have shown that $\lambda = 0.1$ hardly sacrifices completeness where none of the more important MHS solutions are missed, and that virtually the entire probability mass is returned in less than the first $L = 100$ solutions [1]. As a result, STACCATO allows our diagnostic approach to be applied to very large systems.

In terms of the algorithm, STACCATO comprises the following steps:

- Initialization phase, where a ranking of components using the heuristic function borrowed from SFL is computed (lines 1–4 in Algorithm 1);
- Components that are involved in all failed sets are added to D (lines 5–12);
- While $|D| < L$, for the first top λ components in the ranking (including also the ones added to D, lines 13–25) do the following: (1) remove the component j and all A_{i*} for which $e_i = 1 \wedge a_{ij} = 1$ holds from (A, e) (line 17), (2) run STACCATO with the new (A, e), and (3) combine the solutions returned with the component and verify whether it is a minimal hitting set (lines 17–24).

We conclude this section with a complexity analysis of STACCATO. To find a minimal hitting set of cardinality C STACCATO has to be (recursively) invoked C times. Each time STACCATO (1) updates the four counters per component ($O(N \cdot M)$), (2) ranks components in fault likelihood ($O(M \cdot \log M)$), (3) traverses λ components in the ranking ($O(M)$), and (4) checks whether the set is either subsumed or subsumes other sets in D ($O(|D|)$). Hence, the overall time complexity of STACCATO is $O((|D| + M \cdot (N + \log M))^C)$. In practice, however, due to the search focusing heuristic the time complexity is merely $O(C \cdot |D| + C \cdot M \cdot (N + \log M))$. With respect to space complexity, for each invocation of STACCATO the algorithm has to store four counters per component to create the SFL-based ranking ($n_{11}, n_{10}, n_{01}, n_{00}$). As the recursion depth is C to find a solution of the same cardinality, STACCATO has a space complexity of $O(C \cdot M)$. In [1] it has been verified that STACCATO generates solutions with high search efficiency, ordered such that all posterior probability mass is concentrated in the first L solutions. Experiments involving activity matrices of size 30×300 show that diagnostic accuracy is optimal for as low as $L \geqslant 100$. The performance benefits of our approach is exemplified by the fact that matrices with $M = 1\,000\,000$, $N = 1000$, and $C = 1000$ are processed with an average solution rate of 88.6 ms (2.3 GHz Intel Pentium-6 PC with 4 GB memory).

4. Candidate probability computation: BARINEL

In this section we present our approach to compute the g_j and the associated, posterior candidate probabilities $\Pr(d_k)$ given the observations (A, e). In our approach we (1) determine the real g_j instead of $g(d_k)$, and (2) apply the g_j in an improved epsilon policy to compute $\Pr(obs|d_k)$.

The key idea underlying our approach is that for each candidate d_k we compute the g_j for the candidate's faulty components that *maximizes the probability* $\Pr(e|d_k)$ *of the observations e occurring, conditioned on that candidate* d_k (maximum likelihood estimation for naive Bayes classifier d_k).

As in the approximate strategies proposed in previous work, we assume the intermittent failure distributions to be mild, such that even a relatively small amount of observations (N) allow a proper estimation of the g_j. Note, that, unlike the prior p, we do not assume any prior knowledge of g_j. For wild distributions, it may happen that, e.g., for $g = 0.99$ it may take more than $N = 1000$ runs to spot a failure. Consequently, our approach would yield $g = 1$ instead of $g = 0.99$. The mild distribution assumption is acceptable for software, since regression test suites (on which our test data is based) always contain a large number of passing and failing runs, as measured for all activity matrices used in Sections 5 and 6.

For a given process i, in terms of g_j the epsilon policy for (possibly) intermittently failing components is given by

$$\varepsilon = \begin{cases} \prod_{j \in d_k \wedge a_{ij}=1} g_j & \text{if } e_i = 0 \\ 1 - \prod_{j \in d_k \wedge a_{ij}=1} g_j & \text{if } e_i = 1 \end{cases} \tag{6}$$

Eq. (6) reflects the fact that the probability of a process failure is one minus the probability that none of the candidate components induce a failure (g_j per component, the product comes from the failure independence assumption, a common assumption in the diagnosis community).

Algorithm 2 Diagnostic algorithm: BARINEL

Inputs: Activity matrix A, error vector e, number of components M

Output: Diagnostic report D

```
 1: γ ← ϵ
 2: D ← STACCATO((A, e), M, 1, 100)
    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:    ∀_{j∈d_k} g_j ← 0.5
 8:    repeat
 9:       i ← i + 1
10:       for all j ∈ d_k do
11:          g_j ← g_j + γ · ∇expr(g_j)
12:       end for
13:       Pr[d_k]^i ← EVALUATE(expr, ∀_{j∈d_k} g_j)
14:    until |Pr[d_k]^{i−1} − Pr[d_k]^i| ⩽ ξ
15: end for
16: return SORT(D, Pr)
```

In our approach g_j is solved by maximizing $\Pr(e|d_k)$ under the above epsilon policy, according to

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

where $G = \{g_j \mid j \in d_k\}$. Note that for a particular candidate d_k the optimum g_j values may differ with those for another candidate d'_k for the same components.

Our approach, of which the implementation is coined BARINEL, is described in Algorithm 2 and comprises three main phases. In the first phase (line 2) a list of candidates D is computed from (A, e) using STACCATO that returns a list of most probable diagnosis candidates (in our experiments $L = 100$ candidates).

In the second phase $\Pr(d_k|(A, e))$ is computed for each $d_k \in D$ (lines 3 to 15). First, GENERATEPR derives for every candidate d_k the probability $\Pr(e|d_k)$ for the current set of observations e. As an example, suppose the following measurements when c_1, c_2 are at fault (ignoring the healthy components):

c_1	c_2	...	e	$\Pr(e_i\|\{1,2\})$
1	0	...	1	$1 - g_1$
1	1	...	1	$1 - g_1 \cdot g_2$
0	1	...	0	g_2
1	0	...	0	g_1

As the four observations are independent, the probability of obtaining e given $d_k = \{1, 2\}$ equals (Eq. (6))

$$\Pr(e|d_k) = g_1 \cdot g_2 \cdot (1 - g_1) \cdot (1 - g_1 \cdot g_2)$$

Subsequently, all g_j are computed such that they maximize $\Pr(e|d_k)$. To solve the maximization problem we apply a simple gradient ascent procedure [6] (bounded within the domain $0 < g_j < 1$). The motivation behind the choice for a simple, linearly converging, optimization procedure over, e.g., a quadratically convergent, but much more complex procedure, is our focus on demonstrating the added diagnostic accuracy due to our maximum likelihood estimation approach, rather than to minimize computation cost. Moreover, even with the simple optimization scheme, all the test programs are already processed by BARINEL in the order of seconds.

In the third and final phase, the diagnoses are ranked according to $\Pr(d_k|(A, e))$, which is computed by EVALUATE according to the usual, posterior update (Eq. (1)).

For single-fault candidates, the maximum likelihood estimator for g_1 equals the intermittency rate $\sum_i e_i/N$, which is the intuitive way to determine g_1 for single faults. Consider the following (A, e) (only showing the c_1 column and the rows where c_1 is hit), e, and the probability of that occurring (Pr):

c_1	e	$\Pr(e_i\|d_k)$
1	0	g_1
1	0	g_1
1	1	$1 - g_1$
1	0	g_1

As $\Pr(e|\{1\})$ is given by $\Pr(e|\{1\}) = g_1^3 \cdot (1 - g_1)$, the value of g_1 that maximizes $\Pr(e|\{1\})$ is $\frac{3}{4}$, which is easily found by differentiating the expression and determining the zero root. From e we find the same intermittency rate $g_1 = \frac{3}{4}$. While

averaging over e offers a low-cost, analytic solution to computing g_j, this approach only works for single faults, motivating our numeric (gradient ascent) alternative in the multiple-fault case, in contrast to the approximate, averaging techniques ($\varepsilon^{(1,2)}$) published thusfar.

Finally, to illustrate the benefits of our approach, consider the example presented in Section 2.4. As mentioned in the previous section, $\varepsilon^{(0)}$ does not distinguish between candidates with the same cardinality. As (i) candidates rank with the same probability and (ii) the true fault has cardinality 2, all candidates with cardinality 1 and 2 would have to be inspected. $\varepsilon^{(1,2)}$ distinguish between candidates with the same cardinality, but $\{2110, 2745\}$ and $\{2682, 2748\}$ outrank the true fault $\{2553, 2763\}$. BARINEL yields better results due to a better estimation of the individual g_j, ranking the true fault $\{2553, 2763\}$ before all the other diagnosis candidates considered:

d_k	g_j		$\Pr(d_k)$
$\{2553, 2763\}$	$g_{2553} = 0.94$	$g_{2746} = 0.001$	7.8×10^{-4}
$\{2110, 2745\}$	$g_{2110} = 0.26$	$g_{2745} = 0.95$	3.6×10^{-5}
$\{2682, 2745\}$	$g_{2682} = 0.26$	$g_{2745} = 0.15$	9.0×10^{-5}
$\{1347\}$	$g_{1347} = 0.94$		5.8×10^{-20}

As the g_j expressions that need to be maximized are simple and bounded in the $[0, 1]$ domain, the time/space complexity of our approach is identical to the other reasoning policies presented in Section 2 modulo a small, near-constant factor on account of the gradient ascent procedure, which exhibits rapid convergence for all M and C (see Section 6).

5. Theoretical evaluation

In order to assess the performance improvement of our framework we generate synthetic observations based on sample (A, e) generated for various values of N, M, and number of injected faults C (cardinality). 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.

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 C faulty components c_j (without loss of generality we select the first C components) we also set g_j. Thus the probability of a component being involved *and* generating a failure equals $r \cdot (1 - g)$. A row i in A generates an error ($e_i = 1$) if at least 1 of the C components generates a failure. Measurements for a specific (N, M, C, r, g) scenario are averaged over 1000 sample matrices, yielding a coefficient of variance of approximately 0.02.

We compare the accuracy of our approach with previous work in terms of a diagnostic performance metric W, that denotes the excess diagnostic *work* spent in finding the actual components at fault. The metric is an improvement on metrics typically found in the software debugging domain which measure the debugging effort associated with a particular diagnostic method [3,30]. For instance, consider an $M = 5$ component program with the following diagnostic report $D = \langle \{4, 5\}, \{4, 3\}, \{1, 2\} \rangle$ 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 $g_4 = g_5 = 1.0$ the probabilities of the remaining candidates are updated, 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. Consequently, $W = \frac{2}{5}$.

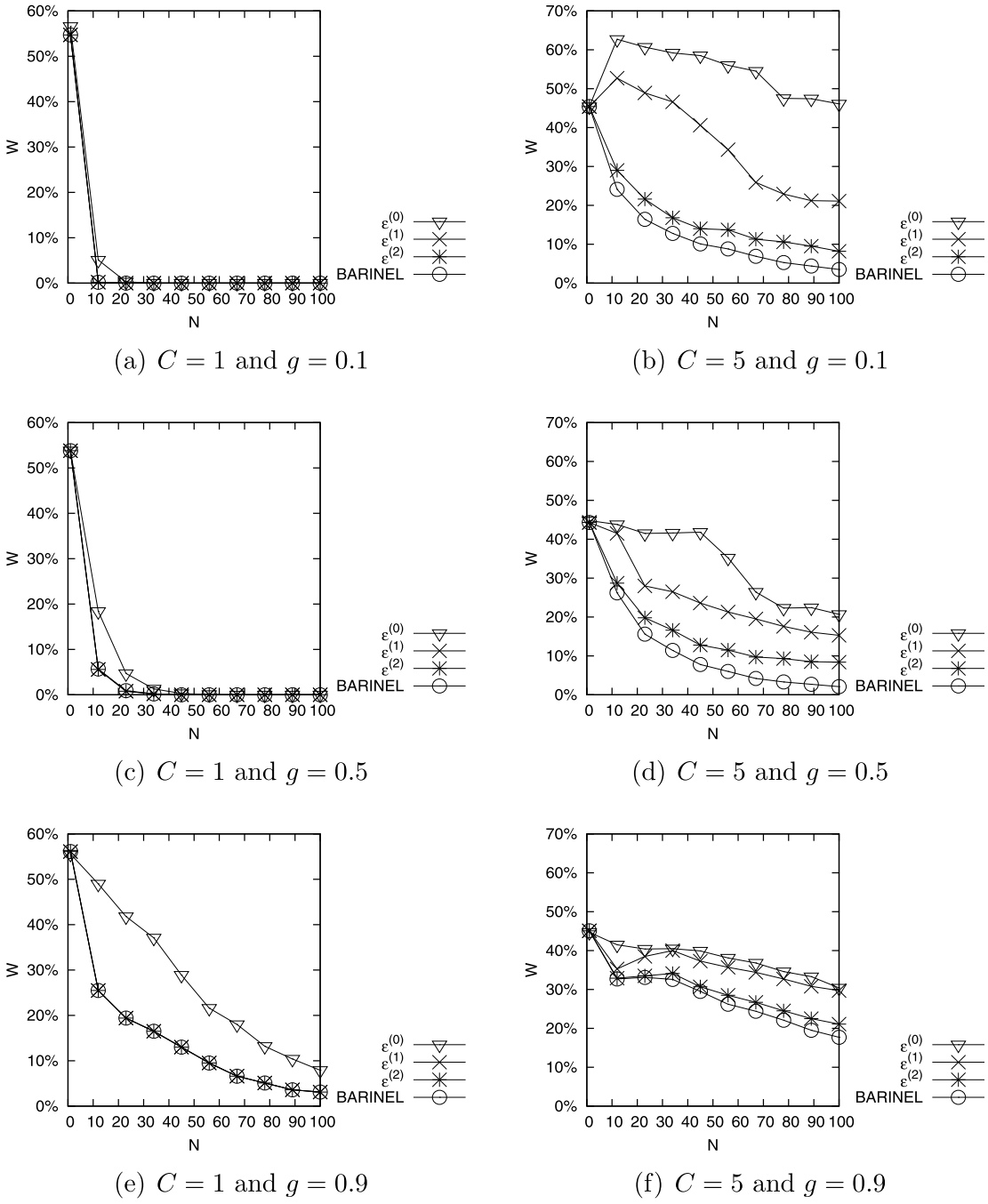
The graphs in Fig. 2 plot W versus N for $M = 20$, $r = 0.6$ (the trends for other M and r values are essentially the same, $r = 0.6$ is typical for real software as found in the Siemens suite), and different values for C and g. The plots show that W for $N = 1$ is similar to r, which corresponds to the fact that there are on average $(M - C) \cdot r$ components which would have to be inspected in vain. For sufficiently large N all policies produce an optimal diagnosis, as the probability that healthy diagnosis candidates are still within the hitting set approaches zero.

For small g_j W converges more quickly than for large g_j as computations involving the faulty components are much more prone to failure, while for large g_j the faulty components behave almost nominally, requiring more observations (larger N) to rank them higher. For increasing C more observations are required (N) 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 $\varepsilon^{(0)}$ is the worst performing policy, mainly due to the fact that it does not distinguish between diagnosis with the same fault cardinality. Only for $C = 1$ the $\varepsilon^{(2)}$ and $\varepsilon^{(1)}$ policies have equal performance to BARINEL, as for this trivial case the approximations for g_j are equal. For $C = 5$ the plots confirm that BARINEL has superior performance, demonstrating that a correct computation of g_j is quite relevant. In particular, the other approaches deteriorate for increasing C.

6. Empirical evaluation

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



(a) $C = 1$ and $g = 0.1$ (b) $C = 5$ and $g = 0.1$

(c) $C = 1$ and $g = 0.5$ (d) $C = 5$ and $g = 0.5$

(e) $C = 1$ and $g = 0.9$ (f) $C = 5$ and $g = 0.9$

Fig. 2. Wasted effort W vs. N for several settings of C and g.

6.1. Experimental setup

For evaluating the performance of our approach we use the well-known Siemens benchmark set [14], as well as the larger programs `space`, `gzip`, and `sed` (obtained from SIR [14]). 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 fault 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. The `Space` package provides 1000 test suites that consist of a random selection of (on average) 150 test cases 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 1000 suites provided.

Table 3
The subject programs.

Program	Faulty versions	M	N	Description
print_tokens	7	539	4130	Lexical analyzer
print_tokens2	10	489	4115	Lexical analyzer
replace	32	507	5542	Pattern recognition
schedule	9	397	2650	Priority scheduler
schedule2	10	299	2710	Priority scheduler
tcas	41	174	1608	Altitude separation
tot_info	23	398	1052	Information measure
space	38	9564	13585	ADL interpreter
gzip-1.3	7	5680	210	Data compression
sed-4.1.5	6	14427	370	Textual manipulator

Table 3 provides more information about the programs used in our 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.

The activity matrices are obtained using the ZOLTAR toolset [21]. 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.

6.2. Performance results

In this section we evaluate the diagnostic capabilities of BARINEL and compare it with other Bayesian policies. Similar to Section 5, 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 4 presents a summary of the diagnostic quality of the different approaches, expressed in terms of wasted debugging effort W (see Section 5). In agreement with the previous section, the results for software systems 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 (1000 sample matrices versus at most 100 matrices in the experiments with real software programs). Only in 4 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 where BARINEL does not clearly outperform the other approaches, and thus the 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 $C = 5$, $\varepsilon^{(2)}$ marginally outperforms BARINEL (by less than 0.5%), which is caused by the fact that (i) the program is almost branch-free and small ($M = 174$) combined with large sampling noise ($\sigma_W/\mu_W = 5\%$ for tcas), and (ii) almost all failing runs involve all faulty components (highly correlated occurrence). For schedule2 with $C = 2$ and $C = 5$, $\varepsilon^{(0)}$ is better due to the fact that almost all failing runs involve all faulty components (highly correlated occurrence). Hence, the program effectively has a single fault spreading over multiple lines, which favors $\varepsilon^{(0)}$ since it ranks candidates with cardinality one first.

6.3. Time/space complexity

In this section we report on the time/space complexity of BARINEL. We measure the time efficiency by conducting our experiments on a 2.3 GHz Intel Pentium-6 PC with 4 GB of memory.

Table 5 summarizes the timing results. The columns show the programs, the average CPU time (in seconds) of BARINEL, $\varepsilon^{(0)}$, $\varepsilon^{(1)}$, and $\varepsilon^{(2)}$, needed to compute the diagnostic report D given (A, e). In all cases, we use STACCATO to generate the candidates. As expected, BARINEL is more expensive than the previous, approximate Bayesian approaches. For example, BARINEL requires about 42 seconds on average for space, whereas $\varepsilon^{(0,1,2)}$ needs less than 1 second. The reason is the numeric, gradient ascent procedure. The effect of the gradient ascent costs is clearly noticeable for the first three programs, as well as space, and is due to a somewhat lower convergence speed as a result of the fact that the h_j are close to 1. Note, however, that the implementation has not been optimized. By using a procedure with quadratic convergence the performance penalty would largely disappear (e.g., 100 iterations instead of 10000, gaining two orders of magnitude of speedup over linear convergence).

In the following we interpret the above cost measurements from a complexity point of view. All techniques update $|D|$ candidate probabilities, where $|D|$ is determined by STACCATO. The complexity of STACCATO is estimated to be $O(N \cdot M)$ (for a constant matrix density r) [1]. Although in all our measurements a constant $|D| = 100$ suffices, 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. For

Table 4
Wasted effort W [%] on combinations of $C = 1$–5 faults for the subject programs.

	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
$\varepsilon^{(0)}$	13.7	18.2	22.8	21.6	26.1	30.8	16.2	25.1	33.8	17.2	23.5	28.6	29.3	**26.6**	**28.9**
$\varepsilon^{(1)}$	1.2	2.4	5.0	4.2	7.6	14.5	3.0	5.2	12.5	0.8	1.6	3.0	22.8	31.4	38.3
$\varepsilon^{(2)}$	11.2	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
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

	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
$\varepsilon^{(0)}$	28.0	26.9	28.7	14.0	18.2	21.5	19.5	25.2	34.3	2.6	5.5	10.2	4.1	6.3	9.3
$\varepsilon^{(1)}$	16.7	24.2	30.5	5.1	8.7	17.4	2.2	3.6	9.5	1.3	2.7	7.8	0.7	0.6	1.8
$\varepsilon^{(2)}$	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
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**

Table 5
Diagnosis cost (time in seconds).

Program	BARINEL	$\mathcal{E}^{(0,1,2)}$
print_tokens	45.3	4.2
print_tokens2	24.7	4.7
replace	9.6	6.2
schedule	4.1	2.5
schedule2	2.9	2.5
tcas	1.5	1.4
tot_info	1.5	1.2
space	41.4	0.9
gzip	28.1	8.2
sed	92	6.7

BARINEL 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 the report is ordered, the time complexity equals $O(N \cdot M + M \cdot \log M)$. The results in Table 5 follow the trends predicted by this complexity analysis.

With respect to space complexity, previous Bayesian approaches need to store the counters $(n_{11}, n_{10}, n_{01}, n_{00})$ used in the probability update per 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 previous, approximate Bayesian approaches. In practice, however, memory consumption is reasonable (e.g., around 5.3 MB for `sed`, the largest program used in our experiments).

7. Related work

As mentioned earlier, in many model-based diagnosis approaches (e.g., GDE [13], GDE+ [31], CDA* [32], SAFARI [15]) failures are assumed to be persistent. Consequently, they may not work optimally when components fail intermittently. Recently, a model for intermittent behavior was introduced as an extension of the GDE framework [9], later extended by [4,5]. As shown by our results, our approach improves on the approximations within these works, providing better results. This paper extends earlier work [2] by (i) including a full description of our MHS algorithm STACCATO, and (ii) including the three, large, real-world programs into the experimental evaluation (`space`, `sed`, `gzip`).

Our approach to diagnosing multiple, intermittently failing defects has been developed and applied in a software fault diagnosis context. In model-based reasoning approaches to automatic software debugging, the model is typically generated from the source code – see [25] for an evaluation of several models. The model is generated by means of static analysis techniques, and is extremely complex. While at this detailed level intermittency is not an issue, the level of detail is such that the associated diagnostic complexity prohibits application to programs larger than a few hundred lines of code. As an indication, the largest program used in [25] is `tcas` (172 lines of code only). In contrast, our low-cost algorithm scales to hundreds of thousands of lines of code. Reasoning approaches based on model checkers include `explain` [19], and Δ-slicing [19], which compare execution traces of correct and failed runs. However, in contrast to our approach, these are not fully automatic as the system under analysis needs to be annotated with pre- and post-conditions to facilitate the generation of the model. In addition, they seem not to scale judging by the fact that the authors have only evaluated these approaches with small programs (only up to 174 lines of code).

Our dynamic approach towards determining component involvement and system failure (i.e., through (A, e)) is inspired by statistical approaches to automatic software debugging, known as spectrum-based fault localization (each row in A is a spectrum). Well-known examples include the Tarantula tool [22], the Nearest Neighbor technique [30], and the Ochiai coefficient [3]. These approaches rank components in terms of the statistical similarity of component involvement and observed program failure behavior. While attractive from complexity-point of view, the approaches do not consider multiple faults. Furthermore, the similarity metric has little value other than for ranking, in contrast to our probability metric.

As for MHS computation, since Reiter [29] showed that diagnoses are MHSs of conflict sets, many (exhaustive) MHS algorithms have been presented in an MBD context. In [18,13,29,33] the hitting set problem is solved using so-called hit-set trees. In [16] the MHS problem is mapped onto a 1/0-integer programming problem. Contrary to our work they do not consider any other information but the conflict sets. It is claimed that the integer programming approach has the potential to solve problems with thousands of variables but no complexity results are presented. In contrast, our low-cost approach can easily handle much larger problems. In [34] a method using set-enumeration trees to derive all minimal conflict sets in the context of model-based diagnosis is presented. The authors merely conclude that this method has an exponential time complexity in the number of elements in the sets (components). The Quine–McCluskey algorithm [28,26], originating from logic optimization, is a method for deriving the prime implicants of a monotone boolean function (a dual problem of the MHS problem). This algorithm is, however, of limited use due to its exponential complexity, which has prompted the development of heuristics such as Espresso (discussed later on).

Many heuristic approaches have been proposed to render MHS computation amenable to large systems. In [24] an approximate method to compute MHSs using genetic algorithms is described. The fitness function used aims at finding

solutions of minimal cardinality, which is not always sufficient for MBD as even solutions with similar cardinality have different posterior probabilities. Their paper does not present a time complexity analysis, but we suspect the cost/completeness trade-off to be worse than for STACCATO. Stochastic algorithms, as discussed in the framework of constraint satisfaction [17] and propositional satisfiability [27], are examples of domain independent approaches to compute MHS. Stochastic algorithms are more efficient than exhaustive methods. The Espresso algorithm [7], primarily used to minimize logic circuits, uses a heuristic to guide the circuit minimization that is inspired by this domain. Due to its efficiency, this algorithm still forms the basis of every logic synthesis tool. Dual to the MHS problem, no prime implicants cost/completeness data is available to allow comparison with STACCATO. To our knowledge the statistics-based heuristic to guide the search for computing MHS solutions has not been presented before. Compared to the above approaches, a unique feature is its heuristic which, given its SFL origin, is specifically tailored to model-based diagnosis.

8. Conclusions and future work

Intermittent failure models can be crucial when modeling complex systems. Estimating the probability that a faulty component exhibits correct behavior is an important step for logic reasoning approaches to properly handle intermittent failures. In contrast to previous work, which merely approximates such probabilities for particular diagnosis candidates, in this paper we present a novel, maximum likelihood estimation approach (BARINEL) to compute the exact probabilities per component at a complexity that is only a constant factor greater than previous approaches due to the use of a heuristic minimal hitting set algorithm (STACCATO) underlying the candidate generation process.

We have compared the diagnostic performance of BARINEL with the classical (Bayesian) reasoning approach, as well as with three reasoning approaches that consider intermittent component failure behavior. Synthetic experiments have confirmed that our approach consistently outperforms the previous approaches, demonstrating the significance of maximum likelihood estimation over approximation. Application to the Siemens benchmark, `gzip`, `sed`, and `space` also suggest BARINEL's superiority (26 wins out of 30 trials), while the exceptions are caused by component clustering in combination with sampling noise.

Future work includes extending the activity matrix from binary to integer, to exploit component involvement frequency data (e.g., program loops), and reducing the cost of gradient ascent by introducing quadratic convergence techniques.

Acknowledgements

We extend our gratitude to Johan de Kleer and Peter Zoeteweij for discussions which have influenced our reasoning approach. Furthermore, we gratefully acknowledge the collaboration with our TRADER project partners. Last but not least, we sincerely thank the reviewers for their invaluable feedback.

References

[1] R. Abreu, 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), AAAI Press, 2009.
[2] R. Abreu, P. Zoeteweij, A.J. van Gemund, A new bayesian approach to multiple intermittent fault diagnosis, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'09), AAAI Press, 2009.
[3] R. Abreu, P. Zoeteweij, 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), IEEE CS, 2007.
[4] R. Abreu, P. Zoeteweij, A.J.C. van Gemund, A dynamic modeling approach to software multiple-fault localization, in: Proceedings of International Workshop on Principles of Diagnosis (DX'08), 2008.
[5] R. Abreu, P. Zoeteweij, A.J.C. van Gemund, An observation-based model for fault localization, in: Proceedings of Workshop on Dynamic Analysis (WODA'08), ACM Press, 2008.
[6] M. Avriel, Nonlinear Programming: Analysis and Methods, Dover, 2003.
[7] R.K. Brayton, A.L. Sangiovanni-Vincentelli, C.T. McMullen, G.D. Hachtel, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Norwell, MA, USA, 1984.
[8] J. de Kleer, Getting the probabilities right for measurement selection, in: Proceedings of International Workshop on Principles of Diagnosis (DX'06), 2006.
[9] J. de Kleer, Diagnosing multiple persistent and intermittent faults, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'09), 2009.
[10] J. de Kleer, Minimum cardinality candidate generation, in: Proceedings of International Workshop on Principles of Diagnosis (DX'09) – Diagnostic Competition, 2009.
[11] J. de Kleer, A.K. Mackworth, R. Reiter, Characterizing diagnoses and systems, Artificial Intelligence 56 (1992) 197–222.
[12] J. de Kleer, B. Price, L. Kuhn, M. Do, R. Zhou, A framework for continuously estimating persistent and intermittent failure probabilities, in: Proceedings of International Workshop on Principles of Diagnosis (DX'08), 2008.
[13] J. de Kleer, B.C. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1) (1987) 97–130.
[14] H. Do, S.G. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact, Empirical Software Engineering: An International Journal 10 (4) (2005) 405–435.
[15] A. Feldman, G. Provan, A.J.C. van Gemund, Computing minimal diagnoses by greedy stochastic search, in: Proceedings of the National Conference on Artificial Intelligence (AAAI'08), 2008.
[16] A. Fijany, F. Vatan, New high performance algorithmic solution for diagnosis problem, in: Proceedings of the 2005 IEEE Aerospace Conference (IEEEAC'05), IEEE CS, 2005.
[17] E.C. Freuder, R. Dechter, M.L. Ginsberg, B. Selman, E.P.K. Tsang, Systematic versus stochastic constraint satisfaction, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'95), AAAI Press, 1995.

[18] R. Greiner, B.A. Smith, R.W. Wilkerson, A correction to the algorithm in Reiter's theory of diagnosis, Artificial Intelligence 41 (1) (1989) 79–88.
[19] A. Groce, Error explanation with distance metrics, in: Proceedings of International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), Springer-Verlag, 2004.
[20] M.J. Harrold, G. Rothermel, R. Wu, L. Yi, An empirical investigation of program spectra, in: Proceedings of International Workshop on Program Analysis for Software Tools and Engineering (PASTE'98), ACM Press, 1998.
[21] T. Janssen, R. Abreu, 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.
[22] J.A. Jones, M.J. Harrold, J. Stasko, Visualization of test information to assist fault localization, in: Proceedings of International Conference on Software Engineering (ICSE'02), IEEE CS, 2002.
[23] L. Kuhn, B. Price, J. de Kleer, M. Do, R. Zhou, Pervasive diagnosis: Integration of active diagnosis into production plans, in: Proceedings of AAAI Conference on Artificial Intelligence (AAAI'08), AAAI Press, 2008.
[24] L. Lin, Y. Jiang, Computing minimal hitting sets with genetic algorithms, in: Proceedings of International Workshop on Principles of Diagnosis (DX'02), 2002.
[25] W. Mayer, M. Stumptner, Evaluating models for model-based debugging, in: Proceedings of International Conference on Automated Software Engineering (ASE'08), ACM Press, 2008.
[26] E.J. McCluskey, Minimization of boolean functions, The Bell System Technical Journal 35 (5) (November 1956) 1417–1444.
[27] M. Qasem, A. Prügel-Bennett, Complexity of max-sat using stochastic algorithms, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO'08), ACM Press, 2008.
[28] W. Quine, A way to simplify truth functions, Amer. Math. Monthly 62 (1955) 627–631.
[29] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1) (April 1987) 57–95.
[30] M. Renieris, S.P. Reiss, Fault localization with nearest neighbor queries, in: Proceedings of International Conference on Automated Software Engineering (ASE'03), IEEE CS, 2003.
[31] P. Struss, O. Dressler, "Physical Negation" – Integrating fault models into the general diagnostic engine, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'89), AAAI Press, 1989.
[32] B.C. Williams, R.J. Ragno, Conflict-directed A* and its role in model-based embedded systems, Discrete Applied Mathematics 155 (12) (2007) 1562–1595.
[33] F. Wotawa, A variant of Reiter's hitting-set algorithm, Information Processing Letters 79 (1) (2001) 45–51.
[34] X. Zhao, D. Ouyang, Improved algorithms for deriving all minimal conflict sets in model-based diagnosis, in: Proceedings of the International Conference on Intelligent Computing (ICIC'07), Springer-Verlag, 2007.