Optimal Discrimination between Transient and Permanent Faults

M. Pizza, L. Strigini
Centre for Software Reliability
City University, London, UK
{mpizza, strigini}@csr.city.ac.uk

A. Bondavalli
CNUCE/CNR
Pisa, Italy
a.bondavalli@cnuce.cnr.it

F. Di Giandomenico
IEI/CNR
Pisa, Italy
digiandomenico@iei.pi.cnr.it

Abstract

An important practical problem in fault diagnosis is discriminating between permanent faults and transient faults. In many computer systems, the majority of errors are due to transient faults. Many heuristic methods have been used for discriminating between transient and permanent faults; however, we have found no previous work stating this decision problem in clear probabilistic terms. We present an optimal procedure for discriminating between transients and permanent faults, based on applying Bayesian inference to the observed events (correct and erroneous results). We describe how the assessed probability that a module is permanently faulty must vary with observed symptoms.

We describe and demonstrate our proposed method on a simple application problem, building the appropriate equations and showing numerical examples. The method can be implemented as a run-time diagnosis algorithm at little computational cost; it can also be used to evaluate any heuristic diagnostic procedure by comparison.

1. Introduction

Fault diagnosis is an important function in computer systems, both for manual repair and for automatic reconfiguration in fault-tolerant systems. A fault may cause the system to violate its requirements, i.e., to fail. Even if the errors caused by a given fault are masked by redundancy, it is desirable to eliminate the fault. Otherwise, additional faults, accumulating over time, may eventually exceed the fault-masking capabilities of the system and cause system failure. Some form of repair or reconfiguration ability is thus generally necessary, except possibly in fault-masking designs for very short mission times.

For the purpose of reconfiguration or repair, a system is subdivided into modules. Those modules that are seen to be faulty are excluded from participating in system operation, as a minimum; in addition, they may be manually repaired, replaced with on-line spares, or their functions may be reallocated to spare modules already present in the system. This action depends on a correct *diagnosis*. Many algorithms exist for automatic diagnosis.

The purpose of diagnosis is often limited to identifying the hardware module(s) affected by faults. However, to choose the right *fault treatment* action it is also necessary to discriminate between permanent and transient faults. With transient faults, modules momentarily become prone to behaving erroneously, though they do not suffer any permanent damage. The natural way of dealing with a transient fault is to keep using the affected module, after recovering any data error caused by the transient fault. In many computer systems, transient faults are known to be the great majority of causes of errors (see, for instance, [11]. Published measures of the ratio between the frequencies of transient and permanent faults vary from 4 to 1000). However, discriminating between transient and permanent faults is difficult. This is the topic of this paper. We will use the term "diagnosis" to designate this specific problem: deciding whether a given module is affected by a permanent fault, a transient fault or no fault.

Discriminating between permanent and transient faults is an important practical problem. In industries like civil aviation and telecommunications, where operating with permanently faulty modules would carry high risks or costs, it is common for modules to be replaced as faulty but later prove to be free from permanent faults, when tested in the repair shop. Treating transient faults as permanent thus has a high cost for these industries. In a system designed for long maintenance-free missions (e.g., a spacecraft), misjudging a module as permanently faulty may cause premature exhaustion of the available stock of spares. On the other hand, continuing to use a permanently faulty module may cause premature failure of the whole system, despite the availability of spares for continued fault-tolerant operation.

Diagnosis is based on the observation of erroneous behaviour. A general description of diagnosis is as follows. A module (which, depending on the specific design, may be as small as a subset of a chip or as large as a whole computer or more) is monitored, either by observing side-effects of its intended computations (typically, signals from error-detection mechanisms) or by applying test procedures every now and then and observing the module's response. Some of the observed behaviours can be diagnosed as erroneous, in the sense that they would not happen unless there is a fault in the system. But a module may behave erroneously for different reasons: a permanent fault, a transient fault or propagation of errors from another module. Decisions about fault treatment should be quite differ-

3rd IEEE High-Assurance Systems Engineering Symposium, Washington, DC, USA, November 1998

ent in the three cases, yet the symptom alone (the erroneous behaviour) is insufficient for deciding among them.

Compared to "ordinary" reliability modelling, reasoning about diagnosis is somewhat counter-intuitive. Reliability modelling predicts the probabilities of certain sequences of events given some assumptions. The modeller plays the role of an omniscient observer. Instead, a diagnostic procedure takes these assumptions, this set of sequences and their probabilities as inputs, but cannot depend on knowing the real sequence under way, and involves questions like "If I observe certain symptoms, which are consistent with more than one of the possible sequences, *which* sequence is really taking place?".

Designers have used many heuristics for discriminating between transient and permanent faults, spanning from simple retry to rather sophisticated off-line error log audit and trend analysis [4, 6]. Heuristics are suggested by intuitive reasoning, and then validated by experiment or modelling (see e.g. [1] for an assessment of a heuristic via modelling). Most on-line techniques [1, 5, 8, 12, 13] use thresholding schemes. They count errors, and when the count crosses a pre-set threshold a permanent fault is assumed.

In contrast to the wide application of heuristics, we have found no previous work stating this decision problem in clear probabilistic terms. Interestingly, a rigorous mathematical formulation of diagnosis is well known. It is based on Bayesian inference [2, 7]. We apply it to discriminating between permanent and transient faults. We refer to an on-line application of the procedure: after each observation (of either erroneous or correct behaviour), our diagnosis procedure runs, instead of a heuristic procedure. It produces an updated probability of the module being affected by a permanent fault. This is an optimal diagnosis algorithm, in the sense that rational fault treatment decisions based on its results would yield the best utility among all alternative decision algorithms using the same information. A rational decision uses the probability of permanent fault to weigh the consequences that any action (e.g., substituting the module) would have if the module were actually permanently faulty vs the consequences if it were not permanently faulty. Indeed, this explicit separation of diagnosis from the subsequent decision is one of the important advantages we seek over the heuristic algorithms, which usually deliver directly a choice of action (or, equivalently, a Boolean decision on considering the module as faulty or not). Diagnosis (the assessed probability of the module being permanently faulty) is affected by certain factors; the utility or loss from an action depend on the state of the module (estimated by diagnosis) but also on other factors (e.g., the mission phase or the amount of spares still available). Hence the desirability of treating the two problems separately.

Bayesian inference gives a standard procedure for an observer who needs to update the probability of a conjecture on the basis of new observations. What is needed is a *prior* probability of the conjecture being true *before* one takes into account the results of the observations. For medical diagnosis, for instance, this probability is typically given by the prevalence of the disease in the population of interest. For hardware diagnosis, this probability is available from the reliability models that designers routinely use, combined with the results of all previous observations.

This approach to diagnosis is applicable in all the circumstances where heuristic methods have been applied. It uses the same information that is used to design and validate them, except that it uses it in a provably correct way according to the axioms of probability. We expect from the Bayesian approach the usual advantages of an exact algorithm over heuristic algorithms. A heuristic algorithm is suitable for a class of applications that must be determined empirically; instead, an algorithm that is provably correct by construction can be trusted to give exact results within a pre-defined scope of application. The Bayesian approach gives a *general* construction method for diagnosis algorithms for any system. We describe its application in a simple case: diagnosis of a single module with exponential reliability.

We give in Section 2 a summary introduction to Bayesian inference, a formalisation of our diagnosis problem, the assumptions we make for our example application and the general form that the diagnosis method takes. Section 3 outlines the behaviour of the algorithm, i.e., how the diagnosis changes as a function of the observed failures, or lack thereof. Section 4 illustrates the behaviour of our diagnosis method and the effects of system parameters via numerical examples. Section 5 discusses our results.

2. Bayesian diagnosis

2.1 Bayesian Inference

Suppose we have a conjecture x, about which we are uncertain. Our degree of belief in the conjecture being true is described by a probability $P(x)$. After observing some new, relevant evidence we want to update our belief in x in a rational way. Both the evidence and the conjecture are described as *events,* that is, subsets of the set of all possible *outcomes* of some *experiment*. Using the definition of conditional probability, one can write that in general:

$P(x|evidence)\ P(evidence) = P(evidence|x)\ P(x).$

Interpreting the left-most probability in this equation as the *posterior* probability of conjecture x (degree of belief in x after observing the new evidence), and the right-most probability as its *prior* probability (degree of belief in x before observing the new evidence), we obtain:

$$P_{post}(x|evidence) = \frac{P_{prior}(x)\ \ P(evidence \mid x)}{P(evidence)} \qquad (1)$$

Given a set C of mutually exclusive conjectures such that their union has probability 1, we can rewrite the right member of the equation and obtain:

$$Ppost(x|evidence)=\frac{P_{prior}(x)\ \ P(evidence\,|\,x)}{\sum_{Conj\in C} P_{prior}(Conj)\,P(evidence\,|\,Conj)}$$

Table 1 lists the events of interest (conjectures and evidence that may be observed) in our context, together with the notation adopted to indicate them, while Table 2 lists the abbreviations used.

correct(i)	the data collected during $observation_i$ are correct
erroneous(i)	the data collected during $observation_i$ are erroneous
success(i)	phase $adjudication_i$ did not detect any error in the data observed
failure(i)	phase $adjudication_i$ detected an error in the data observed
Ok(i)	the module was fault-free during $observation_i$
Perm(i)	the module was permanently faulty during [part of] $observation_i$
Perm	the module is permanently faulty (at a time specified by the context of use)
Trans(i)	the module was not permanently faulty, and suffered from a transient fault or its after-effects (erroneous state), during [part of] $observation_i$
$\neg\, x$	complementary event of event x

Table 1. Definitions of the basic events and symbols used to indicate them

$F_L(0.5)$	number of consecutive failures needed for $P_{post}(Perm)$ to increase from L to 0.5.
L	the limit of $P(Perm)$ after a number of consecutive successes tending to infinity
$P(A)$	probability of event A
$P(A\|B)$	probability of event A conditional on event B
T_{sfr}	"success/failure ratio threshold" (*cf* Sect. 3)
α_p	probability of a permanently faulty module producing correct data when tested
α_t	probability of a module with transient faults (but no permanent fault) producing correct data when tested
$\beta 1$	probability of a module's erroneous response under test being adjudged as correct
$\beta 2$	probability of a module's correct response under test being adjudged as erroneous
θp	probability of a permanent fault occurring between two test rounds
θt	probability of a transient fault occurring between two test rounds
$\lambda p,\ \lambda t$	hazard rates for permanent and for transient faults, respectively

Table 2. Abbreviations used in the analysis

2.2 Test process phases

Let us focus on one module, whose behaviour the designers have decided to monitor. We can model this monitoring as repeated rounds of *testing* of the module. One round is shown in Figure 1. There are two phases. First, in an *observation* phase, a set of data produced by the module is gathered. Then, in an *adjudication* phase, these data are analysed to decide whether they are erroneous or correct. This phase produces the *test result*: *success* if no error has been detected in the data, *failure* otherwise. We shall denote $observation_i$ and $adjudication_i$ the observation and adjudication phase of the i-th test round, respectively. Some examples from current design practice are:

1) modular redundant, voted architectures: the observation phase is the execution of a task up to the production of results to be voted, which are what is observed. The adjudication phase is the voting, which selects a result for output *and* indicates whether the module produced a minority result in the vote;
2) architectures using program code signatures for low-level error detection (cf, for instance, [11]): the observation phase is the period over which a signature is computed, and the adjudication is its comparison with the pre-computed, correct signature;
3) architectures not intended to be fault-tolerant but having some error-detection feature, like memory parity: observations are the operations that may trigger the mechanism, like individual memory accesses, and adjudication is trivial, limited to interpreting a parity error as "failure". This information is seldom used for diagnosis, but it could be by running a diagnosis procedure after each error signal;
4) architectures where modules are periodically taken off-line and tested: the observation is the execution of test programs; the adjudication is the analysis of their results.

We note that the model covers a wide spectrum of situations, and at this stage one can already expect wide ranges for parameter values: the period between test rounds may vary between fractions of microsecond (case 3 above), to milliseconds (case 1) and even months (case 4); the probability that tests of a faulty module will *not* return "failure" also varies by orders of magnitude, being close to 0 in thorough off-line tests but close to 1 in case 3, as most faults will not affect a specific memory access.

2.3 Definitions and assumptions for the examples studied

In terms of Bayesian inference, the result of the test activity is interpreted as the "evidence" in expression (1). At the i-th test round, the set C of possible conjectures has only three elements: *Perm(i)*, *Trans(i)* and *Ok(i)*. The prior probabilities of the conjectures depend on how we model the failure process of the module. We consider the simplest case, that of a module with exponential reliability

(Poisson arrival processes) for both transient and permanent faults. If the test rounds are periodic, the probabilities of the module becoming faulty - with a permanent or a transient fault - in the interval between the ends of two observation phases are two constants, which we call θp and θt, respectively. Then, the prior probability of any conjecture at round i can be obtained from the three probabilities at round i-1, corrected with the probabilities of faults occurring in the meantime. We also consider the module to be *isolated*: there is no propagation of errors from other modules.

For our example we use an additional assumption: while a permanent fault lasts forever, a transient fault (or its after-effects) does not generally continue beyond the beginning of a new observation phase, implying:
$P(erroneous(i) \mid Trans(i\text{-}1))=P(erroneous(i) \mid Ok(i\text{-}1))$;

This assumption simplifies some of our equations: if the effects of a transient fault are observed, they are much more likely to be due to a 'recent' transient fault than to an 'old' one. This is realistic if the duration of transient faults is usually small compared to the time distance between two test rounds, and the effects of transient faults are recovered before the next test round. There are many cases in which this assumption is realistic: with off-line testing, modules are reset before testing; with memory parity, words can be overwritten with correct data before the next test. In many other cases, the assumption will be more or less unrealistic, requiring a more complete model.

From these assumptions and definitions, the following relations follow:

$P(Perm(i+1) \mid Perm(i))=1$

$P(Perm(i+1) \mid Ok(i))= P(Perm(i+1) \mid Trans(i))= \theta p$

$P(Ok(i+1) \mid Perm(i))=0$

$P(Ok(i+1) \mid Ok(i))= P(Ok(i+1) \mid Trans(i))=1\text{-}\theta t\text{-}\theta p$

$P(Trans(i+1) \mid Perm(i))=0$

$P(Trans(i+1) \mid Ok(i))= P(Trans(i+1) \mid Trans(i))= \theta t$

Another set of conditional probabilities is needed:

$P(evidence(i) \mid Conjecture(i))$

where *evidence* is one of {*success, failure*}. This is a set of six parameters, which essentially describe *coverage* factors in:

a) the fault manifestation process, i.e., the way a given fault causes the affected module to produce erroneous -or correct - data during observation phases, and
b) the ability of the adjudication process to discriminate correctly between erroneous and correct data.

We will later model more details of these processes, but first we describe the diagnosis algorithm and give a general idea of how the diagnosis changes with new test results.

2.4 Definition of the method for optimal diagnosis

Using the definitions given, expression (1) for the i-th test round can be written as:

$P_{post}(Conjecture(i))=$

$$\frac{P_{prior}(Conjecture(i))\,P(evidence(i) \mid Conjecture(i))}{\sum_{Conj(i)\in C} P_{prior}(Conj(i))\,P(evidence(i) \mid Conj(i))} \qquad (2)$$

where $Conjecture(i)\in C=\{Ok(i), Perm(i), Trans(i)\}$, and $evidence(i)\in\{success(i), failure(i)\}$

The posterior probabilities P_{post} obtained by applying equation (2) at round *i-1* are then used to derive the prior probabilities to be used in applying (2) to the results of the next round i:

$P_{prior}(Conjecture(i))=$

$$\sum_{Conj(i-1)\in C} P_{post}(Conj(i-1))\,P(Conjecture(i) \mid Conj(i-1))$$

where $C=\{Ok(i\text{-}1), Perm(i\text{-}1), Trans(i\text{-}1)\}$, and the conditional probabilities are as listed in section 2.3. The diagnosis is thus kept up-to-date by repetitively applying equation (2) after each round of testing. At the first round, with i=1, (typically, at the beginning of a mission), however, we need to assign prior probabilities $P_{prior}(Conjecture(1))$ to the three conjectures, *Ok(1)*, *Perm(1)*, *Trans(1)*. For convenience we define a set of probabilities $P_{post}(Conjecture(0))$ to be used to derive $P_{prior}(Conjecture(1))$ at the first round of testing. These probabilities must be chosen to represent the risk that the module is already faulty at this stage. However, we will see that their influence on the diagnosis becomes negligible as new observations accumulate, as is intuitively clear it should do.

This concludes the specification of the diagnosis algorithm. It can be seen that it is computationally cheap. However, this specification does not give a feel for how the algorithm will behave. We know that it infers the correct diagnosis from the test results, in the sense that the diagnosis is consistent with the reliability model from which the detailed algorithm was derived. However, given a specific sequence of observed successes and failures, we would not intuitively know what this diagnosis will be. We will now outline some of the interesting properties of this evolution, and then in Section 4 we will graphically illustrate this behaviour on numerical examples, given specific sequences of test results.

3 Behaviour of the proposed method

Here, we outline how the diagnosis evolves with new evidence, i.e., how the posterior probability of the conjecture "the module is permanently faulty" changes with accumulating test results. A more complete study, which cannot be presented here for lack of space, can be found in [9, 10].

One may first ask what will be the consequences of observing a long series of successes. In this case, the probability of permanent fault asymptotically tends to a limiting value L. Usually, L >0: there is a non-zero probability of the module being permanently faulty, even though it has never failed a test (yet). This may appear counterintuitive, but it can be seen that it is correct, for various reasons: 1) a module that has passed many tests may have become faulty recently, so only the last few tests in the series would have had a chance of detecting the fault; and 2) some faults may be completely undetectable by the tests used, and this would be modelled in the parameter $P(success(i) \mid Perm(i))$. Of course, if there were a permanent fault, a very long series of successes would be unlikely; but if this long series happened, it would not allow one to *completely* rule out a permanent fault.

If instead an uninterrupted series of failures is observed, $P_{post}(Perm)$ increases with and tends to 1 as the number of failures approaches infinity.

All mixed series of failures and successes will cause intermediate behaviours, possibly with oscillations of the value of $P_{post}(Perm)$. The detailed evolution of the diagnosis for any given history of successes and failures can obviously be obtained by repeatedly applying equation (2). However, to understand the influence of the parameters, some less detailed description is desirable. We have defined two such measures, which describe important macroscopic aspects of how $P_{post}(Perm)$ changes with test results, and will be useful in the rest of the paper ([10] gives closed form expressions for these measures):

- a measure of how quickly the diagnosis changes in response to a first failure. In the operation of a system, long series of successes will usually separate any failures or clusters of failures. So, a common case is that a failure is observed at a test round i when $P_{prior}(Perm(i)) \approx L$. A useful descriptive measure is thus $F_L(0.5)$, the number of consecutive failures to be observed for $P_{post}(Perm)$ to increase from L to 0.5. Of course, $F_L(0.5)$ is usually a non-integer. Since test rounds happen in integer numbers, the diagnosis can only be updated after an integer number of observations, and any decision based on it will be somewhat insensitive to small errors in parameter values.
- a measure of how frequently failures must occur for $P_{post}(Perm)$ eventually to tend to 1. When successes and failures alternate (not necessarily in a regular pattern), causing $P_{post}(Perm)$ to oscillate, we would like to know by how much failures must outnumber successes for our diagnosis to veer decidedly towards "permanently faulty". It turns out that this can be determined by reference to a threshold value of the ratio between the numbers of successes and failures, which we call the *success/failure ratio threshold*, or T_{sfr}. If the ratio between the numbers of observed successes and failures asymptotically exceeds T_{sfr}, P_{post}(Perm) decreases towards a value close to L; with a lower ratio, it increases towards 1.

4. Numerical applications

We now take our example of a one-module diagnosis algorithm and instantiate it with parameter values describing two different contexts of application. For each context, we show how the diagnosis would evolve during a specific sequence of test results. We also show how the descriptive measures defined in section 3 capture the macroscopic evolution of the diagnosis over time.

Note that the graphs shown in Figures 2 to 8, like those in previous figures, are not *predictions* of faults and failures, as normally produced in reliability modelling. They describe instead the *diagnosis:* what one can infer about the underlying, invisible fault process, given a specific visible sequence of test results.

Figures 2-8 are intended to be read as "animations" of the test and diagnosis sequence, as follows. Movement towards the right on the x axis represents the flow of time (more precisely, the number of test rounds). Test failures are marked with an arrow. Each number on the x axis (under a test failure) is the number of successes observed since the last failure, as this is a more interesting indication than the number of rounds from the (arbitrary) origin of the time axis. The columns represent the evolution of the diagnosis $P_{post}(Perm)$ with each successive test result. At the beginning of the sequence, $P(Perm)=L$.

But first, we consider the derivation of the parameters for our algorithm.

4.1 Parameters of the diagnosis algorithm - different scenarios

We have specified the diagnosis algorithm in terms of parameters which are conditional probabilities. First, there is the conditional probability of a permanent (resp. transient) fault affecting the module over a certain period of time. Having assumed exponential reliability, this is known given a hazard or arrival rate for the given category of faults and the time duration of interest.

There are then the conditional probabilities of observable test results (success or failure), conditional on the true state of the module. These are determined, in detail, by several factors. We will illustrate their roles via three different, increasingly general scenarios. The evolution of the diagnosis in the three scenarios is studied analytically in [9, 10].

Scenario A ('perfect faults, perfect adjudication'): All faults in a module cause it to fail all tests. So, a test success implies certainty that the module is non-faulty. Clearly, L = 0. The only non-obvious problem in diagnosis is discriminating between transient and permanent faults, after test failures are observed.

Scenario B ('real faults, perfect adjudication'): Even when faulty, a module may still produce correct test data, with probabilities α_p if there are permanent faults and α_t in case of transient faults only. The parameters α_p and α_t describe both the fault manifestation characteristics of the module (e.g., rate of intermittent manifestation of permanent faults) and the coverage of the tests. The adjudication is still 'perfect' (error-free). Observing a success does not warrant certainty that the module is non-faulty. Thus, differently from Scenario A, the limit L is greater than 0.

Scenario C (a realistic scenario) differs from Scenario B in that the adjudication process may misinterpret the data collected, due either to physical faults affecting the adjudication, or to the adjudication being imperfect by design (e.g., software-implemented "reasonableness tests" on program results). This is modelled by 2 parameters: with probability $\beta 1$ erroneous data are interpreted as a success, and with probability $\beta 2$ correct data are interpreted as a failure.

In summary, the simplified Scenario A ('perfect faults, perfect adjudication') is modelled by setting $\alpha_p = \alpha_t = \beta 1 = \beta 2 = 0$. For Scenario B ('real faults, perfect adjudication'), $\alpha_p \neq 0$ and/or $\alpha_t \neq 0$, but $\beta 1 = \beta 2 = 0$

4.2 Example 1: frequent checks on results of application tasks

The module to be diagnosed is a processor or computer, executing a cyclic task at a repetition rate of 20 times per second. The 'Observation' phase is the entire task and the observed data are the execution results, the 'Adjudication' phase takes a negligible time. The processor failure rate for permanent failures is λ_p = 1.2 / 10^5 hours. θp is obtained by multiplying λ_p by the task duration, thus $\theta p = \lambda_p/(20*3600) = 1.7\ 10^{-10}$. We assume that transient faults are 10 times more frequent than permanent faults, i.e., $\theta t = 10*\theta p = 1.7\ 10^{-9}$. Then,

- for Scenario A, $\alpha_p = \alpha_t = \beta 1 = \beta 2 = 0$;
- for Scenario B, we assume $\alpha_p = 0.3$, $\alpha_t = 0.7$, $\beta 1 = \beta 2 = 0$;
- for Scenario C, we keep $\alpha_p = 0.3$, $\alpha_t = 0.7$, and, assuming a small but non-zero probability of adjudication error due to physical faults, $\beta 1 = \beta 2 = 10^{-9}$.

Figure 2 compares Scenario A with Scenario B. In Scenario A, one success is sufficient to make $P_{post}(Perm) = 0$: the memory of previous failures is lost. The plot for Scenario C in Figure 3 is quite different from that for Scenario B in Figure 2, because it assumes $\beta 1$ and $\beta 2$ greater than θp: A test failure is more likely due to adjudication error rather than to an actual fault.

Suppose next, remaining in Scenario C, that transient faults are 100 times more frequent than permanent faults (instead of 10 times). The resulting plot is in Figure 4. An isolated failure now increases $P_{post}(Perm)$ by much less than before: the failure is more likely to have been caused by a transient fault. With reference to Figure 4, it may be useful to point out the information given by the measures $F_L(0.5)$ and T_{sfr}. The first isolated failure in the series of tests increase $P_{post}(Perm)$ less than in Figure 3, as $F_L(0.5)$ is greater. T_{sfr} describes how soon the failures start being 'forgotten'. In Figure 4, $T_{sfr} = 15.43$, and in the sequence of tests there are at least 16 successes for each preceding failure: one can see that there is no consistent upward trend in the values that $P_{post}(Perm)$ takes after each failure. The reader may compare this with the corresponding trends in the preceding figures.

Figure 5 assumes the same parameter values as in Figure 3, except that the adjudication is just a software-implemented reasonableness check on program results, for which we assume low coverage, so that $\beta 1 = P(success(i)|erroneous(i)) = 0.5$. In this case, the second failure (labelled "42") has different effects from those of the first failure (labelled "1"); in Figure 3, the effects of the two were similar. While 42 successes were sufficient to 'reset' $P_{post}(Perm)$ to a value close to L in Figure 3, they are not sufficient in Figure 7, as shown by the fact that $T_{sfr} = 45$.

Returning to the information given by $F_L(0.5)$, we observe that among all these choices of parameters, Scenario B in Figure 2 has the lowest value of $F_L(0.5)$. Thus, $P_{post}(Perm)$ after the first failure is highest in Figure 2. $F_L(0.5)$ in the case of Figure 3 is quite close to the case of Figure 5: after the first failure the value of $P_{post}(Perm)$ is similar for the two cases. Last, one may notice that in all cases $F_L(0.5) < 2$, and two consecutive failures make $P_{post}(Perm)$ very close to 1.

4.3 Example 2: Off-line, periodic testing

Suppose now that, for the same processor as in the previous example, the designer decides to run, instead of error detection concurrent with task execution, a reasonably thorough off-line self-testing program, for one minute in each hour. We thus have $\theta p = 1*\lambda_p = 1.2\ 10^{-5}$. We assume Scenario C. Given the more thorough test procedure, $P(correct(i)|Perm(i))$ is set to a smaller value than in Example 1: $\alpha_p = 0.1$. $\alpha_t = 0.7$ as before. In Figures 6, 7 and 8, transient faults are assumed to be 10, 100 and 1000 times more frequent than permanent faults, respectively, i.e. $\lambda_t = 1.2\ 10^{-4}, 1.2\ 10^{-3}, 1.2\ 10^{-2}$. θt (equal to $\lambda_{t/60}$: only those transient faults that occur during the minute of off-line testing affect test results) thus takes the values: $2\ 10^{-6}$, $2\ 10^{-5}$ and $2\ 10^{-4}$. Since the test lasts longer than in example 1, we assume higher probabilities of physical faults affecting the adjudication phase: $\beta 1 = \beta 2 = 10^{-6}$ in Figures 6 and 7; in Figure 8, instead, the probability $\beta 1$ of adjudication error in case of erroneous data is set to 0.2.

For Figures 6 and 7, $F_L(0.5) < 1$, thus a single failure is sufficient for $Ppost(Perm)$ to become greater than 0.5. The two plots are rather similar in pattern, but $P_{post}(Perm)$ in Figure 7 is always lower than in Figure 6, as the probability that a test failure is due to a transient rather than a permanent fault is higher. Indeed, $F_L\ (0.5)$ is lower for Figure 6.

In Figure 8, transient faults are more likely than in the previous cases, so $P_{post}(Perm)$ increases less after each isolated failure. On the other hand, T_{sfr} is larger, as a consequence of the higher probability of 'false negatives' in adjudication, so more successes are necessary to keep $P_{post}(Perm)$ low. With the sequence of events we have considered, with rather frequent failures, $P_{post}(Perm)$ thus becomes larger than in Figures 6 and 7.

5. Conclusions

We have described an optimal method for discriminating between transient and permanent faults. This is an important practical problem in fault diagnosis, since in many computer systems the majority of errors are due to transient faults. The numerical examples illustrate how the method consistently takes account of intuitively important factors like test coverage and the rates of occurrence of faults. The analytical study explains some aspects of the relationship between the behaviour of the diagnosis algorithm and the reliability model from which it is built.

This Bayesian diagnosis procedure differs from the heuristic methods that are widely used by practitioners in that it separates the phase of computing the probabilities of the possible states of a module from that of deciding a fault treatment action on their basis. Given the diagnosis, i.e., the probability of a permanent fault being present, an optimal fault treatment decision depends on the distribution of the utility (or loss) that will ensue from any combination between a true state and a chosen action: continuing operation with a permanently faulty module, discarding a healthy module, etc.. A common decision criterion (appropriate when the concern is the aggregate utility from many similar systems) is for instance to minimise the expected loss. Calling "Keep" and "Discard" the two possible decisions about a module, and E(decision) the expected loss from a given decision, given a certain true state of the module, the decision algorithm would simply choose to discard a module iff:

$$E(Keep|Perm)P(Perm)+E(Keep|\neg Perm)(1-P(Perm)) > E(Discard|Perm)P(Perm)+E(Discard|\neg Perm)(1-P(Perm))$$

We outlined in the Introduction the other advantages to be expected from applying an exact algorithm, provably correct by construction, in comparison to a heuristic procedure. The amount of improvement that can be achieved depends on the specific application context. However, if the costs of the two approaches were comparable, it would seem natural to choose the exact approach. It is clear from our description that the Bayesian algorithm has limited run-time cost: it amounts to applying equation (2) to update the diagnosis after each new observation, or less frequently, e.g. after each test failure or at preset intervals. Some system parameters need to be estimated to program the algorithm. These are the same parameters that a designer would need to estimate in any case in order to model the consequences of adopting any heuristic decision method. As an alternative to using our method directly, a designer could decide to compare the results of a heuristic decision method against the optimal one. As a minimum, the descriptive measures defined here give an idea of the general trends that diagnosis should follow. They can thus be used to check whether a heuristic procedure that is intuitively appealing will behave correctly on some important reference sequences. They can also be used directly as rough run-time decision rules if more refinement is unnecessary, for instance to set a threshold on the acceptable frequency of failures before a module is treated as permanently faulty.

We have tried to underscore the fact that the Bayesian approach is completely general. Its generality and portability are possibly its main advantages. Given a heuristic algorithm that has been empirically demonstrated to be satisfactory in a given range of circumstances, a designer cannot know whether a similar algorithm will perform well in other circumstances, or even, given these different circumstances, whether *any* simple, satisfactory heuristic can be found for them. Even a change in parameter values (e.g., mission duration or module failure rates) rather than system structure may move the heuristic outside the envelope for which it has been validated. By contrast, a Bayesian algorithm for a new system type is constructed (or its parameters are changed as needed) by applying a standard, direct method.

For reasons of space and simplicity of exposition, we demonstrated the Bayesian approach on the simple example of a single, isolated module. This example may appear contrived to those readers who are familiar with elaborate fault-tolerant design and reliability analyses, yet it is relevant to some everyday problems, like maintenance of stand-alone computers with limited on-line error detection and periodical off-line testing. Designers with a more detailed knowledge (i.e., model) of the behaviour of their systems can accordingly refine the expressions of both the probability of a fault occurring between two test rounds and the probability of detection, false positives, etc. Computing the diagnosis will be more expensive, but it is unlikely to be prohibitively so. For single-module systems, useful refinements to our example can be, depending on the specific circumstances, modelling the distribution, duration and effects of transient faults in more detail, considering the probability of successful recovery, and modelling multiple categories of faults, with different arrival processes and effects.

Among multi-module applications, which we think have immediate interest, there are:

- multiple-redundant, voted systems: the diagnosis algorithm would use the results of adjudication (voting) to diagnose each parallel computation channel, and the diagnosis can be fed back to the voter as explained in [3];
- extending the reliability model that computes the prior probabilities to include error propagation. In a multiple-module system, the diagnosis can be extended to a choice among the possible *combinations* of states (presence of transient or permanent faults) of all modules, thus taking advantage of all the information available to the designer; or it can focus on the individual module of direct interest, representing the influence of the rest of the system simply as a probability of propagated errors.

Acknowledgements

This research was funded in part by the European Commission via the "OLOS" research network (Contract

CHRX-CT94-0577), the ESPRIT Long Term Research Project 20072 "DeVa" (Design for Validation) and the ESPRIT Project 20716 GUARDS (Generic Upgradable Architecture for Real-time Dependable Systems).

References

[1] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico and F. Grandoni, "Discriminating Fault Rate and Persistency to Improve Fault Treatment", in Proc. FTCS-27, Seattle, USA, 1997, pp. 354-362.

[2] M. H. DeGroot, "Probability and Statistics", Reading, Mass, Addison-Wesley, 1986.

[3] F. Di Giandomenico, L. Strigini, "Adjudicators for Diverse-Redundant Components", Proc. 9th Symp. on Reliable Distributed Systems (SRDS-9), Huntsville, Ala., 1990, pp. 114-123.

[4] R. K. Iyer, L. T. Young and P. V. K. Iyer, "Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data", IEEE TC, C-39, pp. 525-537, 1990.

[5] J. H. Lala and L. S. Alger, "Hardware and Software Fault-Tolerance: a Unified Architectural Approach", in Proc. FTCS-18, Tokyo, 1988, pp. 240-245.

[6] T.-T. Y. Lin and D. P. Siewiorek, "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis", IEEE Transactions on Reliability, 39, pp. 419-432, 1990.

[7] H. F. Martz and R. A. Waller, "Bayesian Reliability Analysis", New York, John Wiley & Sons, 1982.

[8] G. Mongardi, "Dependable Computing for Railway Control Systems", in Proc. DCCA-3, Mondello, Italy, 1993, pp. 255-277.

[9] M. Pizza, "Un approccio bayesiano alla diagnosi di guasti hardware transitori e permanenti in sistemi di elaborazione", Thesis, University of Pisa, 1996.

[10] M. Pizza, L. Strigini, A. Bondavalli and F. Di. Giandomenico, "Bayesian Diagnosis of Transient vs Permanent Faults", Centre for Software Reliability Technical Report, 1998. Available from the authors or at: http://www.csr.city.ac.uk/papers/1998.html#pizza_1.

[11] D. P. Siewiorek and R. S. Schwartz, "Reliable Computer Systems Design and Evaluation", Bedford, MA, Digital Press, 1992.

[12] L. Spainhower, J. Isenberg, R. Chillarege and J. Berding, "Design for Fault-Tolerance in System ES/9000 Model 900", in Proc. FTCS-22, Boston, Massachusetts, 1992, pp. 38-47.

[13] N. N. Tendolkar and R. L. Swann, "Automated Diagnostic Methodology for the IBM 3081 Processor Complex", IBM J. Res. Develop., 26, pp. 78-88, 1982.

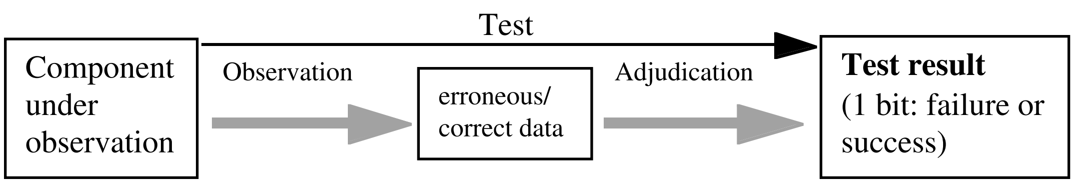

Figure 1. Phases of the test process

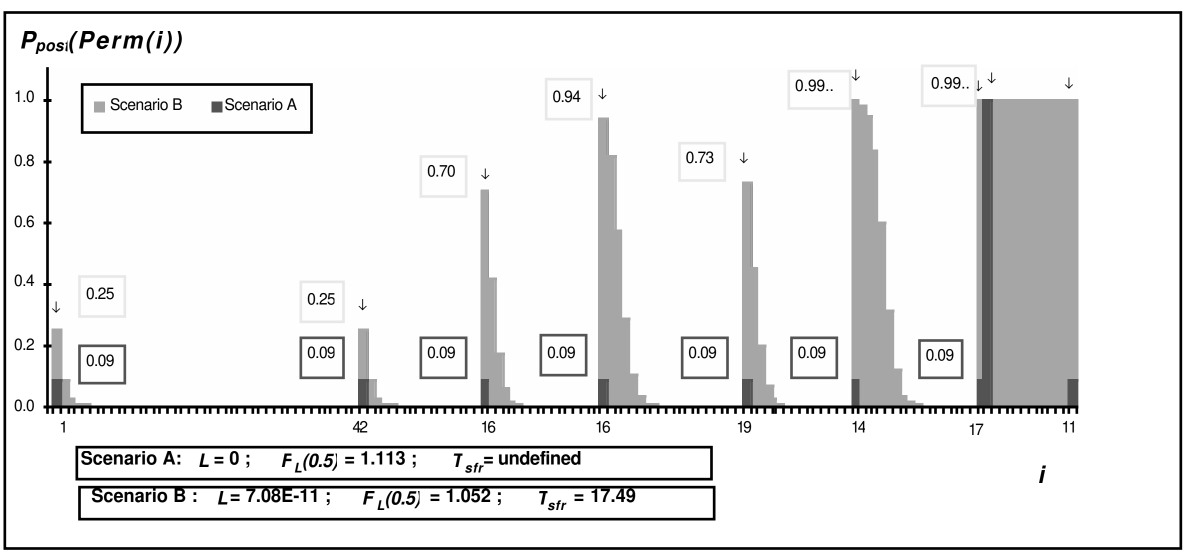

Figure 2. Example 1: different evolution of $P_{post}(Perm)$ for Scenarios A and B, with $\theta t=10^*\theta p$.

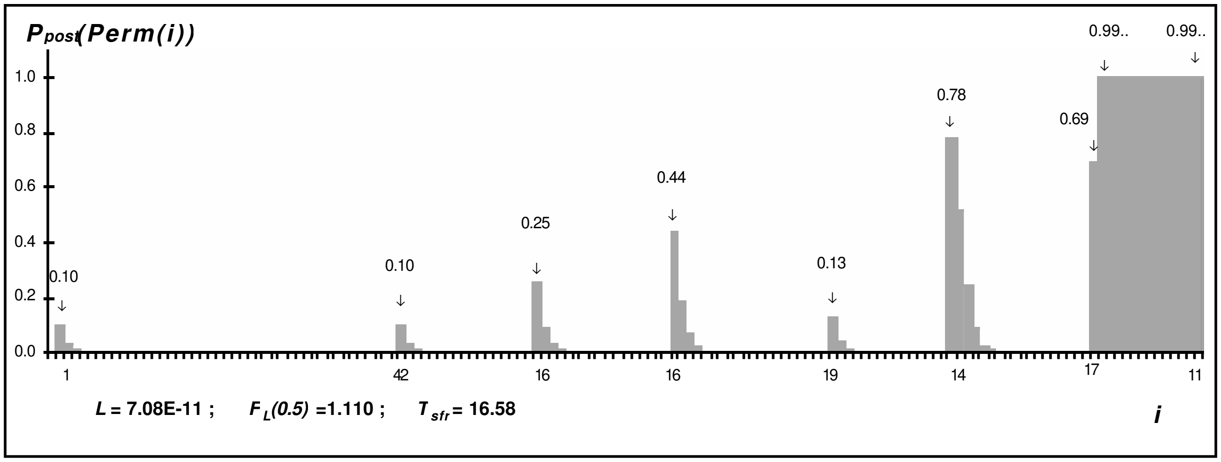

Figure 3. ***P_{post}(Perm)*** **for Scenario C (still with *$\theta t=10^*\theta p$*)**

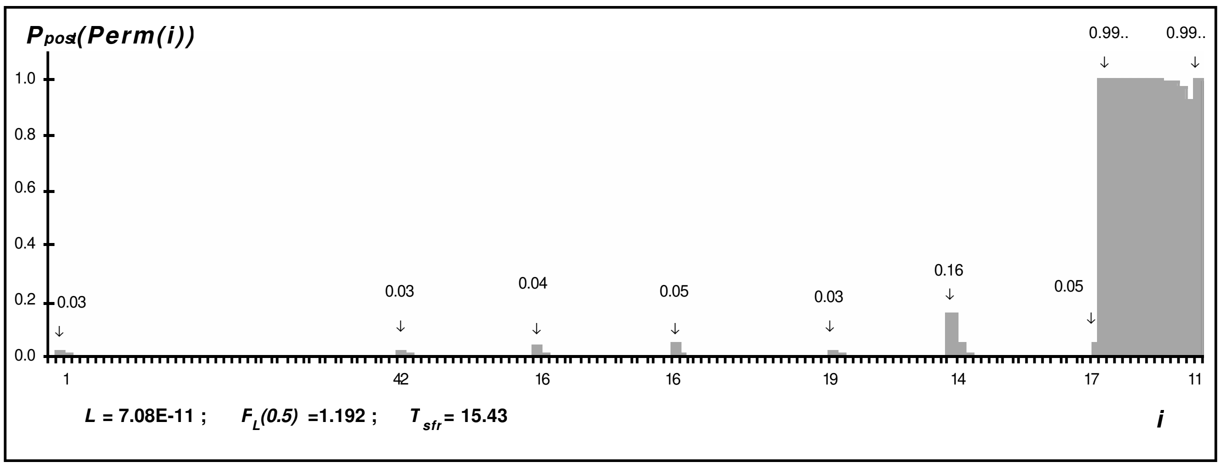

Figure 4. ***P_{post}(Perm)*** **for Scenario C (as in Figure 3, but with *$\theta t=100^*\theta p$*)**

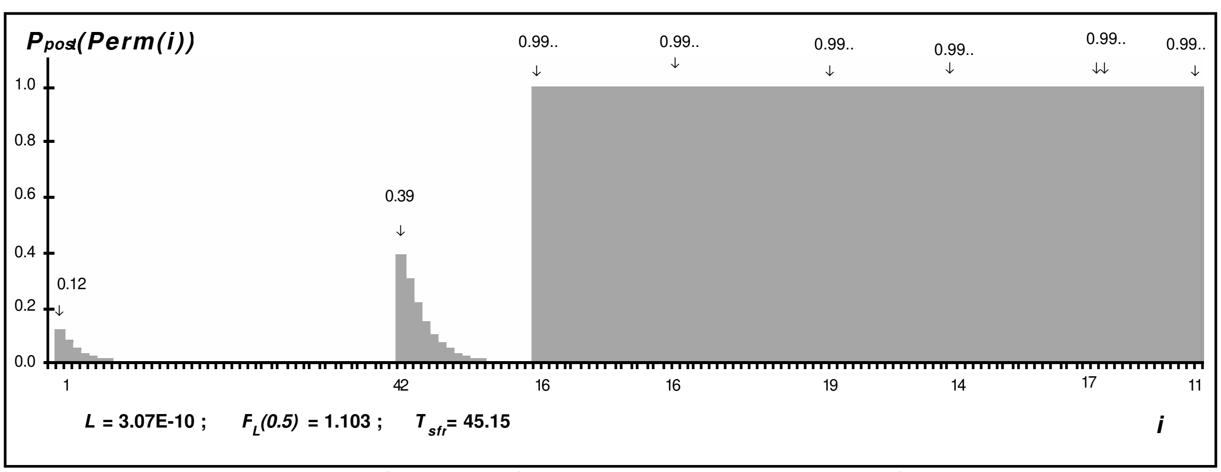

Figure 5. ***P_{post}(Perm)*** **for Scenario C (as in Figure 5, but with *$\beta 1= 0.5$*: acceptance tests with low coverage)**

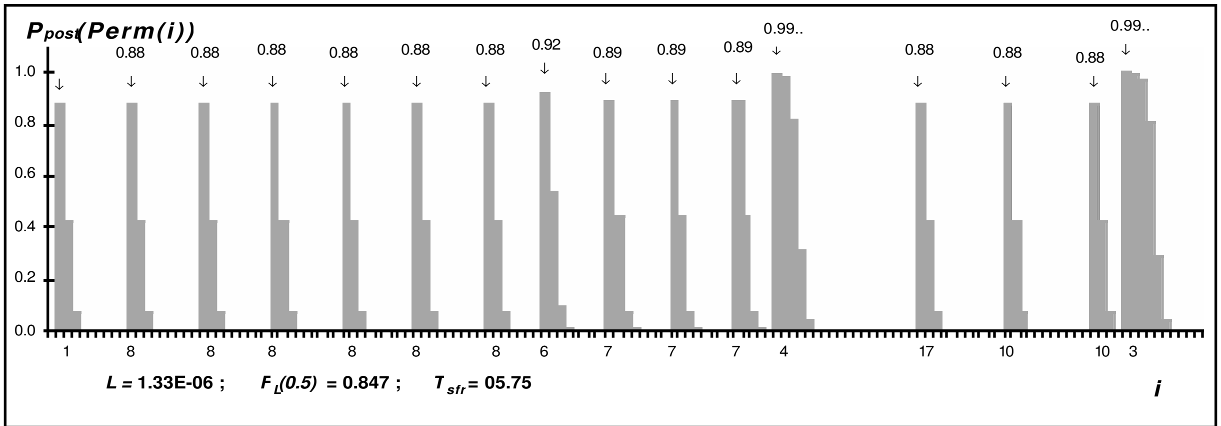

Figure 6. $P_{post}(Perm)$ **for Example 2 (**$\theta t = 2\ 10^{-6}$**,** $\beta 1 = 10^{-6}$**)**

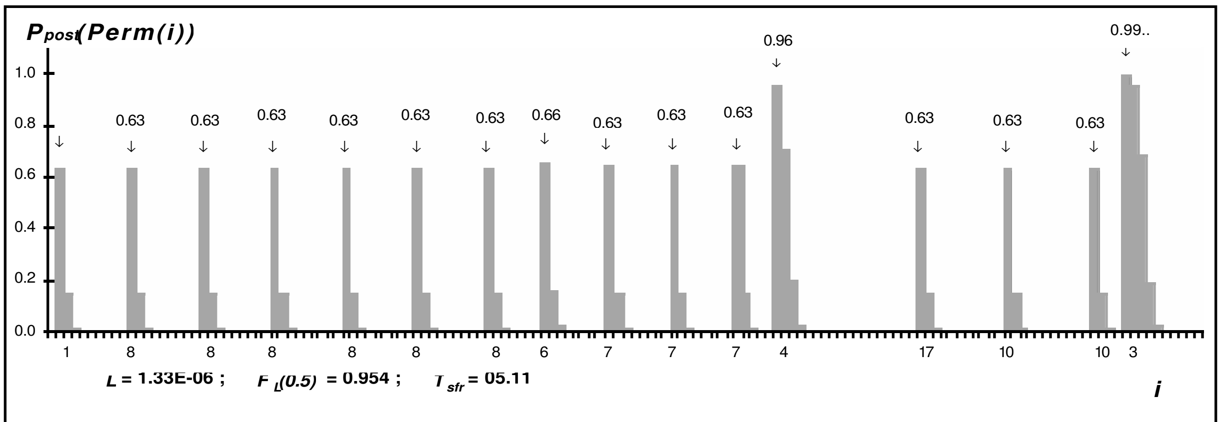

Figure 7. $P_{post}(Perm)$ **(as in Figure 6, but with** $\theta t=2\ 10^{-5}$**,** $\beta 1=10^{-6}$ **)**

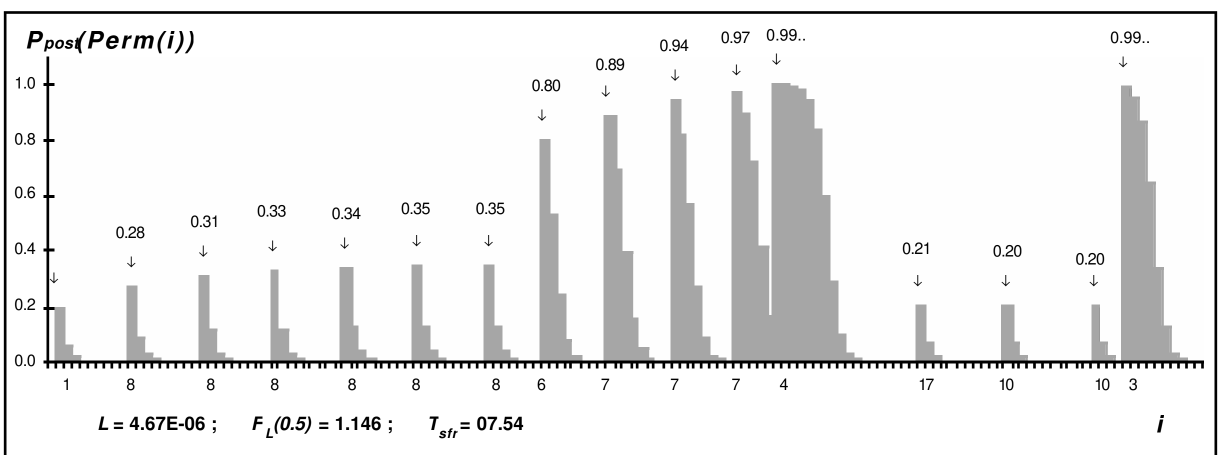

Figure 8. $P_{post}(Perm)$ **(as in Figure 6, but with** $\theta t=2\ 10^{-4}$**,** $\beta 1=0.2$ **)**