Fast Diagnosis With Sensors of Uncertain Quality

Ozgur Erdinc, Craig Brideau, Peter Willett, and T. Kirubarajan

Abstract—This correspondence presents an approach to the detection and isolation of component failures in large-scale systems. In the case of sensors that report at rates of 1 Hz or less, the algorithm can be considered real time. The input is a set of observed test results from multiple sensors, and the algorithm’s main task is to deal with sensor errors. The sensors are assumed to be of threshold test (pass/fail) type, but to be vulnerable to noise, in that occasionally true failures are missed, and likewise, there can be false alarms. These errors are further assumed to be independent conditioned on the system’s diagnostic state. Their probabilities, of missed detection and of false alarm, are not known a priori and must be estimated (ideally along with the accuracies of these estimates) online, within the inference engine. Further, recognizing a practical concern in most real systems, a sparsely instantiated observation vector must not be a problem. The key ingredients to our solution include the multiple-hypothesis tracking philosophy to complexity management, a Beta prior distribution in the sensor errors, and a quickest detection overlay to detect changes in these error rates when the prior is violated. We provide results illustrating performance in terms of both computational needs and error rate, and show its application both as a filter (i.e., used to “clean” sensor reports) and as a standalone state estimator.

Index Terms—Condition monitoring, fault diagnosis, machine health testing, multiphypothesis tracker (MHT), system testing.

I. INTRODUCTION

Recent advances in detection, sensor technology, communications, and computational capabilities have made online system health management an essential and appealing component to large complex system operations. The complexity associated with the maintenance of a large interrelated system such as the space shuttle or a modern aircraft presents formidable challenges to manufacturers and end users. In addition, for redundant (fault-tolerant) systems and for systems with little or no opportunity for repair and maintenance during their operations (e.g., the space station), the assumption of at most a single failure between consecutive maintenance actions is unrealistic. Hence, an efficient maintenance strategy requires computer-aided techniques to design systems for testability, and advanced diagnostic algorithms for troubleshooting.

Online system health management is made possible by smart onboard sensors, where low-level test decisions are made based on the processing of sensed waveforms. A real-time diagnostics algorithm fuses these low-level decisions into an overall assessment of the system state, and a number of algorithms have been proposed for this large-scale fault isolation problem [8], [15], [18], [20]–[22].

Consider a system that consists of 10,000 critical components monitored by 3000 sensors, each of which gives a pass/fail report to the diagnostics system. The diagnostics engine frequently receives reports (every second); however, only a few (on average 5%, or 150) of the 3000 sensors send reports for each scan, meaning that complete information about the station is received, on average, every 20 s. Moreover, the reports are unreliable: a PASS report might come from a broken component, and likewise, a sensor might report FAIL while all the components covered by it properly function. The task of the real-time diagnostics engine is to provide a consistent and reliable system status report by using diagnostics system information and observing such sensor reports. In real-time algorithms, the system is expected to finish processing before a new observation set arrives; hence, the diagnostics engine has to process the data quicker than the scanning rate—within a second in this scenario.

It is desirable for such an algorithm to accommodate test inaccuracies, e.g., missed detections (a “pass” when a failure ought to have been reported) and false alarms (a “fail” reported despite fault-free conditions) in test outcomes. Using the probabilities of missed detections and false alarms in the diagnosis step may improve performance: some knowledge of the reliabilities of the sensors can be used in the diagnosis process itself. In the next section, we shall introduce our solution. It is based on a precise and integrated statistical modeling of the fault isolation and sensor error processes. The probabilities of sensor errors are not known and must be estimated from the data. We impose on them a Beta prior distribution [11]: Beta is appealing from the perspectives of both intuition (a completely uniform prior is a member of the Beta family) and because, largely due to its reproducing structure, the algorithm that arises from it is computationally straightforward.

If the probabilities of false alarms and missed detections were known, the fault process would be a hidden Markov model (HMM) [2]—the unobservable state is the system component fault vector, and the observations are the sensor reports—and an optimal estimator would be available from the Viterbi algorithm [19]. Unfortunately, the stochastic nature of the sensor faults violates the HMM assumptions, since the missed detection and false alarm probabilities are assumed known and constant. Thus, some degree of approximation would be necessary; but in any case, large-scale systems would preclude a Viterbi solution: for example, if there were 100 components (a relatively small system), the number of Viterbi states would be $100^{100}$. In target tracking, a concern is that the “associations” (from measurements to targets) are not known, and hence, in principle, all must be checked. The number of these hypotheses quickly grows large; a solution that tracks only the most promising among them is one of the most favored approaches to target tracking, this being usually known as the multiphypothesis tracker (MHT) [3]. For complexity management purposes, we apply the MHT philosophy to the diagnostic inference problem.

There is some prior literature on diagnosis based on unreliable sensor-level tests [9], [10]. For example, if the reliabilities are unavailable, Hamming distance [19] can be used to find single or multiple faults by comparing the observed test outcome with the test dictionary and selecting the likeliest fault(s) that could have produced the observed sensor outputs. This procedure is fast, but performance is not good, particularly when few sensors are instantiated on a given scan. On the other hand, the most comprehensive approach adopts the same HMM as is used in this correspondence, but relies on the Baum–Welch reestimation to learn—in the maximum-likelihood (ML) sense, jointly with the state—the requisite error probabilities. The resulting procedure performs well, but was found to be very slow, and is probably not suitable for any but small-scale systems. A related and
less computational approach again estimates the missed detection and false alarm probabilities of the sensors online, but bases the estimates only on the ML system state estimate. The procedure is fast, and the performance is much better than for the Hamming match; but the approach is not optimal, and its stability is questionable. In this correspondence, we revisit the problem: via the MHT philosophy and the Beta prior, we develop a statistically sound and computationally fast diagnosis methodology that allows and incorporates the tests’ unreliabilities.

In the following section, we outline the key ingredients for the modeling of the system and the algorithm. In Section III, the maximum a posteriori (MAP) estimation of the system state is derived, and in Section IV, the implementation of this estimation approach is discussed. One application of our algorithm is as intermediary: sensor pass/fail decisions are inherently noisy, and our goal is to reduce this noise before the diagnostics engine interprets them. In a sense, we would provide a preprocessing “filter” to allow the diagnostics engine to deliver its results with confidence. Thus, after the discussion of the algorithm performance in Section V, we describe the implementation of the algorithm when used in cooperation with TEAMS-RT, the diagnostics engine of Qualtech’s Systems’ Integrated Diagnostics Toolset [6].

II. MODELING

A. Dependency Matrix

The Dependency Matrix, or D-matrix, encapsulates the relationship between test observations and the system’s fault condition. In a system where there are M possible faults and N sensors, the D-matrix is an M-by-N matrix for which the \((i,j)\)th element denotes the expected output of sensor \(j\) when fault \(i\) has occurred. If multiple faults occur, the expected noiseless sensor observation would be the or-ed combination of multiple rows of the D-matrix. For our purposes, the D-matrix will be assumed given:\(^2\) to derive it is both complicated and laborious, and probably involves a system expert. For example, a small system may have

\[
D = \begin{bmatrix}
1 & . & . & . \\
1 & 1 & . & . \\
. & 1 & 1 & . \\
. & . & 1 & 1 \\
. & . & . & 1
\end{bmatrix}
\]

as its D-matrix (the dots represent, and are more visually appealing than, “0” D-matrix entries). If fault 2 occurs, then sensors 1 and 4 would indicate a FAIL condition; if, in addition to fault 2, fault 5 occurs, then sensors 1, 4, 5, and 6 would be “lit.” Note that this D-matrix illustrates a fault “masking” condition: for example, if fault 2 occurs, then the additional occurrence of fault 1 cannot be discerned. This has considerable impact on the system’s performance as a state estimator, but it is less of a concern when it is applied as a filter. It should be stated that the general complex systems have D-matrices that are not necessarily square (usually more rows than columns) and have dimensionality into the thousands.

B. Beta Distribution

The probabilities of false alarms and missed detections must be estimated from the data. A natural model to use, since the probabilities are constrained between zero and unity, is the Beta distribution \([14]\). A natural model to use, since the probabilities are constrained between zero and unity, is the Beta distribution \([14]\). The Beta probability density function (pdf) is convenient for the modeling of the binary sensor outcomes, and it integrates estimation of the tests’ inaccuracies with an implicit measure of these estimates’ accuracies. In addition to these advantages, it also results in a feasible and quite efficient algorithm since it is “closed” in the sense that a Beta prior leads to a Beta posterior.

We have

\[
f_{\alpha,\beta}(\alpha) = \frac{(a + b + 1)!}{a!b!} (1 - \alpha)^a \alpha^b
\]

for \(0 \leq \alpha \leq 1\). (Note that \(a\) and \(b\) are integers, and consequently, that we can replace the more general Gamma functions by factorials.) The mean of this pdf is \((a + 1)/a + b + 2\), and as \(a\) or \(b\) increases, the distribution becomes more concentrated around this mean.

C. Integrated Model of System

In the sequel, \(M\) and \(N\) are the numbers of possible faults and tests,\(^3\) respectively. The system state at time \(n\) is \(U_n\), where \(U_n[i] = 1\) means that the fault \(i\) has occurred. The observation at time \(n\) is \(Z_n\), whereas \(O_n[j] = 1\) indicates the event that the \(j\)th sensor actually reports its \(Z_n[j]\) at time \(n\) and \(O_n[j] = 0\) means that it does not.\(^4\) The true sensor output when the state is \(U\) (i.e., what we would observe if the sensors were error free) is \(T(U)\).

As illustrated in Fig. 1, the false alarms and missed detections are defined as follows:

- \(\alpha_j = \Pr(Z_n[j] = 1|O_n[j] = 1, T(U_n)[j] = 0)\): probability of false alarm of sensor \(j\);
- \(\beta_j = \Pr(Z_n[j] = 1|O_n[j] = 1, T(U_n)[j] = 1)\): probability of detection of sensor \(j\);
- \(\nu_i = \Pr(U_n[i] = 1|U_{n-1}[i] = 0)\): probability of failure, per sample period, of fault \(i\);
- \(\theta_i = \Pr(U_n[i] = 0|U_{n-1}[i] = 1)\): probability of self-correction, per sample period, of fault \(i\).

Note that \(Z_n\) may (and generally will) be incomplete, i.e., the outputs from all the sensors may not be available at all times. If only \(K\) tests report, that means \(\sum_{j=1}^{N} O_n[j] = K\).

The \(\alpha\) and \(\beta\)s are accorded Beta distributions with respective parameters \([a, b]\) and \([c, d]\).

The \(\nu\)s and \(\theta\)s cannot realistically be estimated online since it is unlikely that enough fault conditions will occur to allow this. It is reasonable that \(\nu_i\) ought to be set as the reciprocal of the mean time between failures for fault \(i\). A similar value for \(\theta_i\) is reasonable. Usually, a nonextreme prior distribution has little effect, since it is “dominated” by observed data.

\(T(U)\) is derived by or-ing the rows of the D-matrix for which \(U_n[I] = 1\), \(A_{1:m}\) denotes the (generic) ensemble quantity \(\{A_1, A_2, \ldots, A_m\}\).

At most one fault can occur or clear during each scan. That means that the Hamming distance between \(U_n\) and \(U_{n-1}\) is at most unity.

\(^1\)The resulting software was integrated to QSI’s “TEAMS-RT,” but it could have been coupled with any number of inference engine platforms.

\(^2\)The D-matrices used here were mostly specified via QSI’s “TEAMS.”

\(^3\)We exchangeably use the terms “sensors” and “tests” throughout this correspondence. These tests may be complicated test algorithms or, conversely, simple thresholded analog sensors.

\(^4\)The indicator event \(O_n[j]\) relates to the asynchronous nature of the sensors’ reports, and not to communication delays or problems.
The assumption that at most one fault can occur between faults seems strong; yet, if this is not enforced, the ensuing search is of high complexity. However, we can see that if \( M = 3 \) and \( U_n = \{0 \ 1 \ 0\} \), then the only possibilities are \( U_{n+1} = \{0 \ 1 \ 0\}, \{1 \ 1 \ 0\}, \{0 \ 0 \ 0\}, \{0 \ 1 \ 1\}\). That is, the number of transitions to be explored is in general \( M + 1 \) only.

The parameters \( a, b, c, \) and \( d \) of the Beta distributions should be “biased” such that the prior probabilities of false alarm and missed detection are concentrated less than unity. If this is not so, the estimation algorithm may settle for an optimal solution in which false alarms and missed detections are rather more the rule than the exception. We generally set \( a = d = 9 \) and \( b = c = 1 \). Again, note that these are the prior values, and observations deflect them to Beta posteriors that presumably are both more concentrated and better centered on the true values.

### III. Estimation Approach

Next, we will derive the optimal estimator for the state sequence from the beginning up to the current time \( n \), that is, \( U_1:n \). It is desirable to derive a recursive relation so that the algorithm can be implemented by an update equation whenever an observation is received. Our goal is to estimate \( U_1:n \) based on \( Z_1:n \). The criterion for optimality is MAP estimation; that is, we are to maximize the joint pdf \( f(U_1:n, Z_1:n) \).

We have

\[
f(U_1:n, Z_1:n) = f(U_1:n) \prod_{j=1}^{N} \left\{ I(T(U_1) = x)I(Z_1 = y)I(O_1[j] = 1) \right\}
\]

where

\[
\Psi(x, y, j, U_1:n) \equiv \sum_{i=1}^{n} I(T(U_1) = x)I(Z_i = y)I(O_i[j] = 1)
\]

“remembers” the number of observations having a certain accuracy for a given state sequence \( U_1:n \). For example, \( \Psi(0, 1, j, U_1:n) \) recalls the number of false alarms for sensor \( j \). Note that the third element, involving \( O_1[j] \), ensures that only those sensor outputs actually observed are counted.

In (2), the first relation is simply integrating out two variables from a joint density, the second relation is conditioning on \( U_1:n \), and the succeeding manipulations rely on \( f(\vec{\alpha}) \) and \( f(\vec{\beta}) \) being Beta. Assuming that all the sensors’ true missed detection and false alarm probabilities are independent, these are simply products of Beta pdfs. In the last relation, the integrals are replaced by the inverse of the Beta coefficients given in (1), since each pdf has to integrate to unity.

The goal of the MAP estimator is to maximize \( f(U_1:n, Z_1:n) \) over the state sequence \( U_1:n \). Using the above equation together with the assumption that \( U_1:n \) is a Markov process, the log likelihood of a particular sequence \( U_1:n \) can be written as

\[
\log \{ f(U_1:n, Z_1:n) \} = \log \{ f(U_1:n−1, Z_1:n−1) \} + \log \{ f(U_n|U_{n−1}) \}
\]

where we have (5), shown on the bottom of the page.

As it is clear from (4), the optimal MAP estimation can recursively be accomplished as the new observations arrive, resulting in a Viterbi-like trellis [19]. That is, each state \( U_{n−1} \) can be propagated to its \( M + 1 \) possible descendants at scan \( n \), with a log-likelihood cost evaluated according to the above equations. Then, for each \( U_n \), the ML is selected, and the corresponding counts \( \{\{\Psi(x, y, j, U_1:n)\}_{x=0}^{1}\}_{y=0}^{1} \}_{j=1}^{N} \) accordingly updated (see Fig. 4). The fact that the MAP estimation procedure has this form is a key strength of the Beta prior pdf on the false alarm and missed detection probabilities.

However, there is a concern. It is true that each “state” at scan \( n−1 \) must be updated, although given the single-change-per-scan assumption, the cardinality of the updates is only \( M + 1 \). However, the number of possible states is unreasonably large: there are (in theory) \( 2^{MN} \) states, and if we take \( M = 1000 \) as a reasonable size, this is clearly impractical.
In the next section, we describe the novel complexity management philosophy, namely, the multiple-hypothesis tracking (MHT) approach.

IV. ALGORITHM

The fact that the false alarm and missed detection rates are unknown precludes a simple HMM approach. This problem often arises in target tracking problems where an optimal estimator would be available if all of the parameters of the system under study were known prior to calculation. Based on the similarities between our problem and target tracking, we suggest the MHT [3], [16], [17]. In target tracking, the number of “hypotheses”—in the form of target/measurement associations—exponentially increases with time. MHT is a rubric describing means by which the number of “hypotheses” can be controlled to a manageable number.

One way to graft a feasible MHT to this problem is to calculate the likelihoods of all possible states that are one component different from the current state. As illustrated in Fig. 2, when there are Top-K kept states from the previous iteration and M components in the system, the Top-K*(M + 1) states need to be searched to find the most likely. We will call this implementation the “standard version” of the algorithm (Fig. 3).

No practical MHT can be considered an optimal target tracking approach. Similarly, our MHT-grounded Top-K scheme is not optimal for state estimation. The MHT works well, however; we expect that as long as Top-K is large enough so that the true hypothesis remains somewhere in the kept set, our algorithm’s performance will be close to optimal.

Although MHT makes this algorithm feasible, when M is large (e.g., thousands), the computational burden can still be undesirably high. For such cases, we modify the standard version to search through fewer hypotheses. The assumption of at most one new fault per scan allowed us to reduce our search M + 1 states for each “kept” hypothesis (Section II-C). However, we can reduce this further by using the information from current observation: if there is no sensor failure reported related to a fault, we do not include that fault in the current search list. In this approach, there are three subsets of states that need to be searched for each of the previous kept states:

1) the previous kept state itself;
2) the states that have one fewer failed components than the previous kept state (self-correction);
3) the states that contain one more failed components that are supported by the current observation.

The first two subsets are obvious choices of states (inherited from the standard version) to check. In the first, we are checking the likelihood that the state has not changed since the previous iteration. In the second, we are checking the possibility that one of the components that has not been properly functioning in the previous iteration now is. The third hypotheses (states) subset is more complicated to see than the previous two and can be explained in steps as follows (Fig. 4).

- **New FAIL search**: The algorithm checks if there are new failing sensors in the incoming observation that were not marked as failed in the cumulative observation vector, i.e., the full-length vector that holds the most recent report from each sensor.
- **Corresponding new candidate components search**: If the New_FAIL vector is not empty, a search through the New_FAIL [i]th column of the D-matrix is made, and the rows having a “1” in this column are determined. In other words, the components that possibly cause those sensors to fail are determined and are listed as New Candidate components.
- **Formation of hypotheses**: By using each component of the New Candidate list, the neighbor states—that are no more than unity Hamming distance away from the previously survived states—are formed.

Through experimentation, we have seen that this propagation scheme is up to 30 times faster than the propagation to the all-neighbor states scheme. Table I shows the performance of the previous filter version (Top-K*(M + 1) complexity) versus the current version in one particular simulation of the “Documatch” model, to be described later. Clearly, the speed-up is almost an order of magnitude, while the relative increase in errors is significant, the absolute error rate remains low. However, in applications that speed is not critical, the standard version of the algorithm might be chosen for its lower error rate.

The faster algorithm can be particularly useful in cases where there is a large number of components that are reported on by only a few sensors per scan (i.e., a sparsely instantiated observation vector); this is a common situation, particularly if the sampling period is short (a few seconds). Since little information becomes available in each observation, the “fast version” may explore considerably fewer states than the “standard.”
In choosing the $k$ (“Top-K”) states to store for the next iteration, some checks have to be performed on the states that are kept. Certainly, the states have to be checked to make sure that the state is not being saved more than once. Perhaps more subtly, the fault signatures (i.e., the observation vectors) corresponding to the kept states must be checked. It is possible to have an identical fault signature for different combinations of rows of D-matrix, since masking is common in real systems. The algorithm prevents the keeping of redundant states that cannot be disentangled, and, hence, frees up space to explore a wider variety of hypotheses.

A. Page’s Test and Quickest Detection

As explained in Section II-B, the probabilities of detection and false alarm are estimated via Beta distribution modeling for each sensor. However, a practical feature needed in the estimation is the ability to allow the false alarm and/or missed detection probabilities to change, a kind of “forgetting factor.” Without this, as the algorithm would run, it would become increasingly certain of its false alarm and missed detection probabilities, and if these probabilities were to drift from their estimated values for some reason, the result could be a “runaway” condition.

Page’s test [12], [13] seems appropriate: it is the quickest way to see when observations have switched from being governed by pdf $f_0$ to pdf $f_1$. In Page’s test, the “cusum” statistic

\[ S_0 = 0 \]

\[ S_n = \max \{ 0, S_{n-1} + g(U_n) \} \tag{7} \]

is calculated, and each time it exceeds a threshold $h$, a detection is declared. In the preceding equation, $S_n$ corresponds to the cusum statistic at time $n$ that is tested against a threshold, and $g(U_n)$ is (optimally) the log-likelihood ratio of the two hypotheses [i.e., the tested hypothesis $f_1(u)$ and the null (current belief) hypothesis $f_0(u)$] expressed as

\[ g(U_n) = \log \left( \frac{f_1(u)}{f_0(u)} \right). \tag{8} \]

Now, we do not know $f_1$; therefore, it seems reasonable to work with the four possibilities that each sensor probability jumps by a factor $\alpha$; that is, we have

\[ f_0 = P_f^{2(1-T)}, (1 - P_f) \begin{pmatrix} 1 - (1 - (1 - T))^z \end{pmatrix}, Pr(1 - (1 - T))^{ZT} \cdot (1 - P_m)^{ZT} \tag{9} \]

\[ f_1 = \alpha P_f^{2(1-T)}, (1 - \alpha P_f) \begin{pmatrix} 1 - (1 - (1 - T))^z \end{pmatrix}, Pr(1 - (1 - T))^{ZT} \cdot (1 - P_m)^{ZT} \tag{10} \]

\[ f_1b = \frac{1}{\alpha} P_f^{2(1-T)}, \begin{pmatrix} 1 - (1 - (1 - T))^z \end{pmatrix}, Pr(1 - (1 - T))^{ZT} \cdot (1 - P_m)^{ZT} \tag{11} \]

\[ f_1c = P_f^{2(1-T)}, (1 - P_f) \begin{pmatrix} 1 - (1 - (1 - T))^z \end{pmatrix}, \alpha Pr(1 - (1 - T))^{ZT} \cdot (1 - \alpha P_m)^{ZT} \tag{12} \]

\[ f_1d = \frac{1}{\alpha} P_f^{2(1-T)}, (1 - P_f) \begin{pmatrix} 1 - (1 - (1 - T))^z \end{pmatrix}, \frac{1}{\alpha} Pr(1 - (1 - T))^{ZT} \cdot (1 - \frac{1}{\alpha} P_m)^{ZT}. \tag{13} \]

Here, $Z$ corresponds to a “1” in the observation, and $T$ corresponds to a “1” in the perceived truth determined by the filter. Notice that at each sensor observation, only one of the four terms is nonunity in each of (9)–(13). In our implementation, we use $\alpha = 1, 2$.

When a detection occurs for a sensor, i.e., the cusum statistics $S_n$ exceeds a threshold, we simply set the estimator parameters to the initial values, i.e., $\alpha = 1$ and $b = 9$. More complicated reinitialization schemes can be found in the literature with an expense of extra memory requirements [1].

V. RESULTS

A. Monte Carlo Simulations

Data are generated starting with a perfectly functioning system (i.e., no component failures) and adding failures as observations come along. A new observation arrives every 2–3 s, and the system encounters a new faulty component, on average, every 30 scans, i.e., 60–90 s. We first randomly create the system’s status and then calculate the corresponding observation via the OR-ing operation.

The probability that a component currently passes but fails in the next scan is $0.03*1/M$, and the probability of self-correction of a component is $0.5*1/M$ at each scan, where $M$ is the number of components in the system. These numbers were chosen so that, on average, every 30 scans a new component goes bad, but such that the number of bad components does not rise over 6% of the number of components in the system. When there are so many failed components, it is virtually impossible to recover since there are too many sensors firing to determine when components become good (self-correct) again. This seems reasonable, since the diagnosed system will likely be repaired when there are many failed components.

In the following, we use four system models to investigate the algorithm. Three are real.

Documatch 259 components, 180 sensors; LGCU 2080 components, 1319 sensors; X38 594 components, 560 sensors; Simulated model 600 components, 400 sensors.

Documatch is a TEAMS D-matrix representation of the Pitney Bowes Integrated Mail System that takes an original document from a Windows-based personal computer and turns it into a finished properly addressed matched mail piece in a sealed envelope. LGCU-WRA is a control unit for the landing gear system of a Sikorsky S92 commercial helicopter. It includes a hydraulic system that controls the action of the gear and sensors that monitor the status (flying/grounding) of the helicopter. Finally, the X-38 is a model of one of the wiring harnesses in the National Aeronautics and Space Administration’s X-38 aircraft. The simulated model uses a randomly generated D-matrix of size 600 by 400 with sparsity 5%.

Recalling from Section II-A that the D-matrix consists of “0’s and “1”s, the mean values of rows and columns (which are less than or equal to unity) of a D-matrix give a better understanding about the underlying system. These values are calculated by summing and normalizing all the entries in each row (in Fig. 5) and in each column (in Fig. 6) of the D-matrix.

The most immediate observation from the figures is that the random model is rather different from the real ones. The number of sensors attached to each component (histogram of rows) is randomly spread with the mean value 0.05 (sparsity percentage). The same distribution is observed in the number of components each sensor is connected to (histogram of columns). Real models also have interesting structures; X38 has perhaps the most desirable structure, in that the column-wise and row-wise mean values mostly do not exceed 0.01. This implies a

\[ \text{This rate is considerably higher than expected in real systems; however, to measure the algorithm’s performance, we need faults.} \]

\[ \text{We interchangeably use the term component with fault, assuming every component induces only one fault.} \]
Fig. 5. Component-wise histograms of the four models. For example, approximately 140 components of the Documatch system have mean 0.0111, meaning that each of these 140 components is covered by exactly two sensors.

Fig. 6. Sensor-wise histograms of the four models. For example, approximately 270 sensors of the LGCU system have mean value close to 0.9, meaning that each of these 270 sensors covers more than 1800 components.

more “inferable” system, since each component is monitored by only a few relatively unique sensors.

Documatch has a similar structure, and its column-wise mean values do not exceed 0.04, which implies it has sensors connected to only a few components, i.e., ten sensors are each connected to 0.027*594 = 16 components (Fig. 6). LGCU is a much more integrated system: its sparsity is around 20%. This can directly be seen from the row-wise histogram in Fig. 5. However, its column-wise histogram is interesting: a considerable number of sensors (approximately 270) have 0.9 mean values—these sensors cover 90% of the whole component set of the system. It is more likely that “masking” occurs in LGCU than for the other models.

In the following, we used 10000 scans for each run. The probability of detection $P_d$ changes between 90% and 99.9%, and the false alarm rates are in the range of 0% to 5%. The complexity parameter Top-K versus error rate is plotted in Fig. 7.

Increasing the Top-K in the range of 1–5 dramatically decreases the error rate, as intuitively expected. It is interesting that the rate of improvement decreases beyond this point, and the gain from the increasing complexity of the algorithm is negligible, except the Simulated Model, and as discussed before, this is related to the random nature of the D-matrix. Real models are far less “random,” since they are constructed by deliberately assigning groups of sensors to cover sets of failures: subject to a permutation, the D-matrix may appear more block diagonal. Considering that real models’ performances are of primary interest, fixing Top-K at 5 appears to be a good choice. This value will be used in the sequel to explore the effect of test availability.

The error rate for filtering is calculated by comparing the filtered output—that is, the sensor signature that would be observed if the most likely state among the Top-K were true and the sensors were noise-free—to the true noise-free sensor data that are, fortunately, available within the simulation. It is important to note that the error rate reported is computed by comparing filtered and true values only for those sensors that report during a given scan.

It is expected that as the test availability rate increases, the error rate would decrease. There are two effects at work here: the delay in correcting the system state estimate due to sensor reports that have simply not been observed, and a more serious kind of error when the delay of a particular report causes the true state to “fall” from the Top-K kept system states. We would expect the latter to be catastrophic, but it appears from Fig. 8 that even with very low test availabilities (2.5%), only the former effect dominates.

The computational load (processing time per scan, measured on a 2.8-GHz Pentium-IV machine and using the C language) is illustrated in Figs. 9 and 10. It is expected that the load should increase directly proportional to Top-K, and this is indeed observed. It is perhaps less obvious that a lower test availability would result in a speedier
algorithm, since a lower number of reporting tests does translate to fewer successor states that need to be explored. In fact, this efficiency arises from a parsimonious observation data structure. At any rate, observe that the numerical load is not high (Fig. 10). A comparable result from [9] is that the fast (and suboptimal) Hamming and ML techniques run in 65–100 ms on a 300-MHz machine.

Fig. 11 shows the effect of implementing Page’s test in the filter algorithm. Here, a drastic change in a particular sensor’s behavior is simulated. During the first thousand scans, the probability of false alarm is set at 2%. This value is then changed to 30% for the next thousand scans. Because of the nature of the Beta distributions used for the false alarm and missed detection parameters, once a sensor has enough data to become sure of its false alarm rate, it is difficult to drastically change even when a drastic change occurs in the system. The dotted line in Fig. 11 shows the slow increase in the estimated false alarm rate for this sensor. Page’s test, as described earlier, allows the drastic change to happen as soon as—a couple of scans—a change is detected. Observe in Fig. 11 that the solid line, output of the estimator with Page’s test implemented, has a steep slope once the change is detected at scan 1183. Note that Fig. 11 refers to two different Monte Carlo runs, so that a slight difference in the estimated values is observed until scan 1183 due to statistical noise.

The effect of Page’s test is dramatic. If it was not implemented, during the period of the slow increase in the Beta pdf’s estimation, the error in the belief of the false alarm rate would result in an error in state estimation. Consequently, this would cause a loss in the performance of the algorithm.

B. Filter Implementation of the Algorithm

The algorithm is integrated into the TEAMS structure as a preprocessing block before the inference engine TEAMS-RT receives observations. We have employed two TEAMS-RTs: one receives sensor data directly, and the other is fed by a filter output and makes the inference with this ”clean” data. The performance difference is shown in Fig. 12. In this Monte Carlo run, the real model “X-38” is used with 100% test availability and Top K is 15. The false alarm rate of each sensor is randomly chosen between 1% and 20%, whereas the missed detection probabilities are between 10% and 20%. On average, every 150 scans, a new fault is added to the system state, which results in six faults at the end of the run.

False alarms and missed detections of sensor observations naturally cause flickering errors in inference (topmost plot in Fig. 12). While false alarms cause wrong “Bad” inferences, missed detections for a failing sensor clear the components fail. By using the filter, these errors are totally eliminated except for a few cases in which inference is temporarily wrong. Note that these errors disappear within the next scanned data.

The proposed algorithm cleans the observations via its estimate of the system state (Section III). Thus, it is an inference engine itself. In the bottom plot of Fig. 12, the performance of the algorithm as a state estimate can be seen. Its performance is equivalent to that of TEAMS-RT, except for a few cases. Note that this representation does not report “Suspect” components (between scans 540 and 600, TEAMS-RT reports three components as “Suspect,” whereas our inference is wrong for two components).

Another important effect of using the filter is the vast decrease in the number of suspects at the inference. Each false alarm brings a considerable amount of faults into the suspect category since a failing sensor has to be taken as evidence of a fault in the inference mechanism. When this evidence is not strong enough to declare a fault as “Bad,” the inference mechanism reports all the related faults as “Suspect,” i.e., all components in the failing sensor’s column are declared as “Suspect.” This effect causes, on average, 60 components to be in the “Suspect” list. As shown in Fig. 13, the filter eliminates this effect, and the number of components in the inference drops down to reasonable limits for a correct inference (between 0 and 3 for this particular run). In the inference, the suspects are mainly due to ambiguities of the system model. This can also be explained by a “masking” situation of the D-matrix, as discussed in Section II-A. In the bottom plot of Fig. 13, the peaks are the temporary inference of TEAMS, which
at most stays until the next scan, and are perhaps due to techniques used in TEAMS to improve the real-time processing capability of the engine [7]. Although this effect is undesirable in general, it does not harm human interpretation.

VI. SUMMARY

We treat the problem of machine diagnostic state inference with noisy binary sensors whose probabilities of false alarm and missed detection are unknown and must be estimated from the observations. We show that the problem can be posed as one very similar to data association, a key ingredient to target tracking; lessons learned from target tracking may be valuable. We have developed an algorithm whose speed and performance are good. One of its key features is a Beta prior distribution on the individual sensor fault processes. Another key feature of the algorithm is its complexity management: using a philosophy similar to the MHT, it prunes the “kept” hypotheses down to a relevant few.

Some of the important aspects of the algorithm are as follows.

1) It is primarily designed to perform as a “filter,” and it is generic in the sense that it can be used in conjunction with any diagnostics engine.
2) In such an application, even if the observation vector is partially reported in each scan, the algorithm can provide a full observation vector—based on its state estimate—to the diagnostic engine.

3) The complexity management is governed by design parameters. If desired, the number of “kept” hypotheses can be chosen large, and the algorithm works close to the optimally; otherwise, a lower value can be chosen for greater speed but some loss in accuracy.

4) It can be used as a standalone inference engine.

The performance of the filter is presented, along with a comparison between its state estimate and the corresponding TEAMS-RT’s inference.

There is some concern that a nonfailure-caused drift sensor false alarm and missed detection probabilities may be catastrophic. We have incorporated Page’s test to detect such drifts and readjust the algorithm to rapidly estimate the new probabilities.

ACKNOWLEDGMENT

The authors would like to thank the valuable assistance of Dr. S. Deb and V. Malepati from QSI.

REFERENCES


