Diagnosis for Uncertain, Dynamic and Hybrid Domains using Bayesian Networks and Arithmetic Circuits

Brian Ricks
Ole J Mengshoel
Diagnosis for uncertain, dynamic and hybrid domains using Bayesian networks and arithmetic circuits

Brian Ricks *, Ole J. Mengshoel *
Carnegie Mellon University, NASA Research Park, Moffett Field, CA 94035, USA

A R T I C L E   I N F O
Article history:
Received 22 February 2013
Received in revised form 11 February 2014
Accepted 13 February 2014
Available online 18 February 2014

Keywords:
Bayesian networks
Arithmetic circuits
Diagnosis
Hybrid
ProDiagnose
CUSUM

A B S T R A C T
System failures, for example in electrical power systems, can have catastrophic impact on human life and high-cost missions. Due to an electrical fire in Swissair flight 111 on September 2, 1998, all 229 passengers and crew on board sadly lost their lives. A battery failure most likely took place on the Mars Global Surveyor, which unfortunately last communicated with Earth and thus ended its mission on November 2, 2006. Fault diagnosis techniques that seek to hinder similar accidents in the future are being developed in this article. We present comprehensive fault diagnosis methods for dynamic and hybrid domains with uncertainty, and validate them using electrical power system data. Our approach relies on the use of Bayesian networks, which model the electrical power system, compiled to arithmetic circuits. We handle in an integrated way varying fault dynamics (both persistent and intermittent faults), fault progression (both abrupt and drift faults), and fault behavior cardinality (both discrete and continuous behaviors). Our work has resulted in a software system for fault diagnosis, ProDiagnose, that has been the top performer in three of the four international diagnostics competitions in which it participated. In this paper we comprehensively present our methods as well as novel and extensive experimental results on data from a NASA electrical power system.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Historically, electrical power systems (EPSs) have been a major safety concern for people on-board aircraft and spacecraft, and have also caused damage to or loss of unmanned aircraft and spacecraft. Examples of aircraft accidents and incidents include those of Swissair flight 111 and an EasyJet Airbus A319. Because of an electrical fire in Swissair flight 111 on September 2, 1998, all 229 passengers and crew on board tragically lost their lives. On September 15, 2006, an EasyJet Airbus A319 suffered an EPS failure due to a misdiagnosed intermittent fault. Fortunately, the A319 incident did not result in loss of human life or vehicle. An example of an EPS accident on a spacecraft involves the Mars Global Surveyor, where a battery failure most likely took place. The spacecraft last communicated with Earth on November 2, 2006, and NASA officially ended its mission in January 2007.

More generally, a broad range of faults can take place in EPSs of vehicles, and their dynamic and continuous nature often plays a significant role. For example, the dynamics of intermittent and drift faults often make them hard to diagnose. Furthermore, diagnostic systems designed to monitor EPSs often operate in the presence of very little sensed environmental data. To compound these issues, a faulty sensor may give incorrect or extremely noisy readings.

* Corresponding authors.
E-mail addresses: brian.ricks@sv.cmu.edu (B. Ricks), ole.mengshoel@svc.cmu.edu (O.J. Mengshoel).

http://dx.doi.org/10.1016/j.ijar.2014.02.005
0888-613X/© 2014 Elsevier Inc. All rights reserved.
Fault diagnosis techniques that seek to hinder or reduce the impact of similar future accidents and incidents are being developed in this article. Our work integrates probabilistic computations into diagnosis algorithms [1–5], using Bayesian networks (BNs) [6,7]. BNs provide a solid foundation for a diagnostic system to efficiently compute correct diagnoses when provided incomplete, uncertain, or noisy information or data, for example EPS sensor data. Based on the types of random variables Bayesian networks contain, we can partition BNs into three classes: discrete BNs (with discrete random variables only); continuous BNs (with continuous random variables only); and hybrid BNs (with both discrete and continuous random variables).

Even though it may appear natural to use hybrid BNs in hybrid domains such as EPSs, there are also limitations associated with doing so. The mathematics of hybrid BNs is non-trivial, thus the need to introduce restrictions such as linear Gaussians [8] or resort to approximations [9]. In addition, arithmetic circuits, which we compile our BNs to for purposes of embedded and real-time computing, do not currently support continuous or hybrid BNs.

This article discusses our approach to diagnosis in the presence of incomplete and uncertain information, using BNs [6,7] and arithmetic circuits [10–12] for computation. Our BN models are generated by an algorithm which takes as input a schematic of the physical system to be modeled [13]. These static BN models have been augmented, as discussed in this article, to better support hybrid (in particular, continuous) and dynamic behavior [3,5,14,15]. To meet the challenge of handling hybrid and dynamic behavior using discrete and static BNs, we have developed several novel techniques. Some of these techniques are based on the computation of cumulative sum (CUSUM), a sequential analysis method [16]. We discuss in this article how to use CUSUM as a method for bringing out time-dependent and continuous fault behavior in EPSs, and using CUSUM results as evidence in our static and discrete BNs.

Our diagnostic algorithm ProDiagnose takes data from an environment (for example an EPS) and performs real-time diagnostics on the data. We demonstrate our software implementation of ProDiagnose by applying it to data from a real-world EPS called ADAPT [17], located at the NASA Ames Research Center. ADAPT resembles EPSs found in aircraft and spacecraft, and consists of three redundant power sources connected via relays and circuit breakers to two load banks. Also incorporated into ADAPT are different types of sensors that measure various phenomena that reflect the behavior of the EPS's components.

In experiments reported in this article, we show strong diagnostic performance using ProDiagnose, both in terms of accuracy and computational speed. ProDiagnose accuracy results were consistently above 85%, with computation times between 300 and 400 microseconds. Also, the quality of our results did not dramatically drop when ProDiagnose was applied to newer datasets from the ADAPT electrical power system. This shows the potential robustness of our approach to the natural process of electrical power system aging.

ProDiagnose had the best performance in three out of four international diagnostic competitions, arranged at the 20th and 21st International Workshop on Principles of Diagnosis (DX-09 and DX-10). The competitions were organized in 2009 and 2010, respectively, and partly focused on ADAPT. In addition to experimental results for the datasets used in these two competitions, we present in this article also results for the 2011 competition, in which ProDiagnose did not compete.

This work is focused on the problem of fault diagnosis in dynamic and hybrid settings, using static and discrete BNs. Unlike some previous work, our use of one health node per component and sensor allows us to avoid the much simplifying single fault assumption. Discrete BNs have previously been used for fault diagnosis in terrestrial EPSs [1,18], although not for the broad range of faults, including abrupt continuous faults, that we stress here. A benefit of using static rather than dynamic BNs is the reduction in model development time as well as computational effort. A benefit of using discrete rather than hybrid Bayesian networks is that we can compile discrete BNs to arithmetic circuits, taking advantage of research advances for arithmetic circuits [10–12].

This article improves and expands upon previous workshop and conference papers [3,5,14,15]. We integrate and tie these previous papers together into a more coherent and comprehensive picture compared to what has been provided earlier. This article presents the ProDiagnose system in its entirety, and provides additional results and details that further explain and validate our models, algorithms, and software. We present results from new experiments, using data from the competition arranged as part of the 22nd International Workshop on Principles of Diagnosis (DX-11). Our novel experimental results reflect the effect of the aging of ADAPT on ProDiagnose’s performance as well as sub-millisecond computation time for diagnosis.

The rest of this article is organized as follows. In Section 2 we briefly review Bayesian networks and arithmetic circuits, related work in diagnosis, and introduce the ADAPT EPS. We discuss in Section 4, using case studies, how and why electrical power systems fail and introduce various fault types found in these systems. Section 5 explains our approach to constructing a Bayesian network model of an electrical power system, and Section 6 presents our diagnosis framework and algorithms, which we call ProDiagnose. Section 7 discusses experimental results, while Section 8 concludes and outlines areas of future research.[3]

---

1 See https://sites.google.com/site/dxcompetition2011/ for more information on these competitions, and https://c3.nasa.gov/dashlink/static/media/publication/iijphm-dxc.pdf for a description of the evaluation methods used.

2 Under the single fault assumption, it is for simplicity assumed that an EPS can only exhibit one fault at a time; multiple concurrent faults are assumed to not happen.

3 This article is based on previous workshop/conference papers, more specifically [3,5,14,15].
2. Preliminaries

In this section we introduce Bayesian networks and arithmetic circuits as well as previous work on hybrid systems diagnosis. We identify two areas of related work on hybrid systems: research using Bayesian networks, and research using other techniques. We discuss previous work in both areas, with a particular emphasis on research on fault diagnosis in electrical power systems.

2.1. Bayesian networks and arithmetic circuits

Diagnostic problems can be solved by taking a model-based approach, in which one creates a relatively detailed mathematical model of the system of interest. We investigate diagnostic models represented as random variables structured using directed acyclic graphs (DAGs); this approach is called Bayesian networks (BNs) [6,7,19,20]. The vertices of these graphs, which we will refer to as nodes, represent discrete random variables. Each node takes on a finite set of values, called states. The edges of a Bayesian network represent dependencies between nodes, and are often structured to represent causal relationships. Distributions for each node are represented as conditional probability tables (CPTs). Let \( X \) represent the set of all nodes in a BN, \( \Omega(X) = \{x_1, \ldots, x_m\} \) the states of a node \( X \in X \), and \( |X| = |\Omega(X)| = m \) the node's cardinality (number of states). Further, let \( Pa(X) \) represent the parents of a node \( X \in X \). A CPT consists of a node's states as the rows, and parent states as columns. Thus, the size of a node's CPT is dependent on \( m \) as well as the cardinality of its parents, where the latter determines the number of CPT columns. Specifically, the number of columns is given by \( \prod_{i=1}^{n} |Pa(X)| \), where \( m = |\Omega(Pa(X))| \).

By taking a subset \( E \subseteq X \), denoted the evidence nodes, and clamping each of these nodes to a specific state, the answers to various probabilistic queries can be computed. Formally, we are providing evidence \( e \) to all nodes \( E \), where \( E = \{E_1, E_2, \ldots, E_n\} \), \( e = \{E_1(e_1), E_2(e_2), \ldots, E_n(e_n)\} \) and \( e_i \in \Omega(E_i) \) for \( 1 \leq i \leq n \) and \( n \leq m \). Probabilistic queries for BNs include the marginal posterior distribution over one node \( X \in X \), denoted BEL\((X, e)\), over a set of nodes \( X \), denoted BEL\((X, e)\), and most probable explanations (MPES) over nodes \( X \rightarrow E \), denoted MPE\((e)\).

While Bayesian networks can be used directly for inference [6,21,22], in this paper we compile them to arithmetic circuits (ACs) [10,12,23], which are then used to answer BEL and MPE queries. An arithmetic circuit derives marginal probabilities by addition and multiplication operations, and inference is performed using an arithmetic circuit evaluator (ACE) [10,12]. During each call to ACE, the partial derivatives of an AC are computed with respect to each discrete random variable. The arithmetic circuit computes marginal probabilities in time constant in the size of the circuit.

Key advantages to using ACs are speed and predictability, which are important for resource-bounded real-time computing systems including those found in aircraft and spacecraft [2,5,13,14,24–26]. Compared to alternative approaches to computation using BNs, for example join tree clustering [19] and variable elimination [22], AC computation often has substantial advantages in terms of speed and query accuracy, even when implemented in software as done in this paper and previously [10,12–14]. The fundamental limitation of ACs is that they may, in the general case, grow to the point where memory is exhausted. This may become a problem in highly connected BNs. The BNs investigated in this paper, as well as in similar fault diagnosis applications, are typically quite sparse [3–5,13,14], and memory consumption turns out not to be a problem as reflected in Section 7.

2.2. Diagnosis using Bayesian networks

From the perspective of Bayesian model-based diagnosis, Lucas distinguishes between Bayesian abductive diagnosis and Bayesian consistency-based diagnosis [27]. Our work takes, similar to Pearl [6], an abductive approach to model-based diagnosis and uses Bayesian networks. Early similar work on BN diagnosis formalized the Quick Medical Reference (QMR) knowledge base [28]. Other early application areas include robotics [29] and fall diagnosis [30].

Based on clique tree propagation [19], Olesen developed an approach to exactly compute marginals in clique trees that are compiled from hybrid BNs [8]. In order to maintain exactness, the hybrid BNs were restricted to ones in which the continuous nodes have continuous children. For a continuous node, each discrete configuration of parents gives a linear Gaussian distribution. For each configuration of all discrete nodes, the continuous distribution is multivariate Gaussian, and this approach therefore is a generalization of mixtures of Gaussians.

Koller and Lerner also investigated hybrid BNs, and introduced an inexact particle filtering approach for computing marginals [9]. Let \( X(t) \setminus E(t) \) represent the set of non-evidence nodes at time \( t \) (slice) \( t \). Each particle is an instantiation of \( X(t) \setminus E(t) \), and the belief state at time \( t \) is approximated by all particles. This particle filtering algorithm approximates marginals at time \( t \) over all non-evidence continuous and discrete nodes.

Diagnosis of electrical power systems using BNs has been explored previously [1,13,18,31]; this includes work on BN auto-generation from a high-level specification language [13,31]. There is also related work on sensor validation [32,33], which can be considered as diagnosis of sensors. Software health management, including diagnosis, has also been performed by means of BNs [25,26,33].
Table 1
A breakdown of sensor quantity for the ADAPT electrical power system. Information for the configuration of DP2 and DP1 is shown. The Bayesian network sensor quantities describe the quantity of nodes for each network structure in the BN representations of DP2 and DP1 (see Section 5.1 and Section 5.2).

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Description</th>
<th>ADAPT Qty of sensor per EPS</th>
<th>Bayesian network Qty per sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>DP2</td>
<td>DP1</td>
</tr>
<tr>
<td>DC Current Sensor</td>
<td>it</td>
<td>Measures DC current in amps.</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>AC Current Sensor</td>
<td>it</td>
<td>Measures AC current in amps.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>DC Voltage Sensor</td>
<td>e</td>
<td>Measures DC voltage in volts.</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>AC Voltage Sensor</td>
<td>e</td>
<td>Measures AC voltage in volts.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Circuit Breaker</td>
<td>ish</td>
<td>Senses the position (open or closed) of a circuit breaker.</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Position Sensor</td>
<td>esh</td>
<td>Senses the position (open or closed) of a relay.</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>te</td>
<td>Measures temperature in Fahrenheit.</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Speed Transmitter</td>
<td>st</td>
<td>Measures the RPM of fan blades.</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Flow Transmitter</td>
<td>ft</td>
<td>Measures the flow rate of a pump in gallons per hour.</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

2.3. Diagnosis using other approaches

Among research using other techniques, we consider first model-based fault diagnosis in hybrid systems [34,35]. Narasimhan and Biswas developed a model-based diagnosis approach based on hybrid Bond graphs [34]. The approach integrates tracking (using an extended Kalman filter), fault detection (which compares estimated and observed signals), fault isolation, and fault identification. Successful experimental results are shown for a fuel-transfer system of a fighter aircraft. Narasimhan and Mengshoel integrated BNs and consistency-based diagnosis [36]. The work by Daigle et al. is also based on hybrid Bond graphs [35]. This research is similar to ours in its emphasis on electrical power systems and ADAPT. Specifically, it also deals with abrupt continuous and discrete faults. Unlike our research, this work makes the single fault assumption [35]. RODON, a model-based approach [37] based on the general diagnostic engine [38], also participated in DXC09 with good results [39–41].

An optimization-based approach to fault diagnosis has been applied to ADAPT as well [42]. This approach, which obtained strong results in DXC09, amounts to developing a linear model of the EPS circuit, and diagnosis is then based on solving a convex problem that includes faults and other hidden states.

3. The ADAPT electrical power system

The Advanced Diagnostics and Prognostics Testbed (ADAPT) is a real-world electrical power system (EPS) located at the NASA Ames Research Center. ADAPT is similar to electrical power systems found aboard spacecraft and aircraft, while providing a controlled environment for injecting faults and recording data [17]. ADAPT was the testbed used to generate the datasets for the 2009–2011 DX Competitions (called DXC in this article). These were international competitions which evaluate entrants’ diagnostic algorithms against one another.5

In the DXC10 and DXC11 competitions, ADAPT had two configurations, DP1 and DP2, as shown in Fig. 1. Table 1 contains information about sensors in ADAPT, for both DP1 and DP2 configurations. Also listed in Table 1 is a breakdown of Bayesian network node quantities for each sensor type (number of nodes per sensor type) in the BN models of DP1 and DP2. How these nodes connect to each other to represent network structures of physical sensors is discussed in Section 5.1 and Section 5.2.

The DP2 configuration represents the ADAPT EPS in its entirety. This configuration consists of three batteries connected to two load banks [17]. Each load bank can handle both AC loads and DC loads; the former are connected to the distribution network through a DC–AC inverter.

The ADAPT DP1 configuration, as shown as the green shaded region in Fig. 1, represents a subset of DP2. The DP1 configuration consists of a single battery connected to a subset of the second load bank, which consists of 2 AC loads (connected through the lower DC–AC inverter) and a single DC load.

While this article focuses on the DP1 and DP2 configurations of DXC10 and DXC11 (Fig. 1 and Table 1), prior work [3,14] used the DP1 and DP2 configurations of DXC09. There are two main differences between the DXC09 configurations and the DXC10 and DXC11 configurations:

---

5 See https://sites.google.com/site/dxcompetition2011/ for more information on these competitions, and https://c3.nasa.gov/dashlink/static/media/publication/ijphm-dxc.pdf for a description of the evaluation methods used.
• DP2 in the DXC09 configuration (also called Tier 2 in [3,14]) contained more sensors than the DXC10 and DXC11 configurations (Fig. 1 and Table 1). This produced more complete environmental data, which was deemed unrepresentative of real-world EPSs on-board aircraft and spacecraft. Consequently, ADAPT DP2 was stripped of many sensors to generate the DXC10 and DXC11 datasets.

Fig. 1. The ADAPT electrical power system (EPS). This is the schematic after which the Bayesian network shown in Fig. 11 (Section 5.3) is modeled. The entire EPS is also referred to as DP2. The green shaded region denotes the ADAPT DP1 subset (configuration). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
• DP1 in the DXC09 configuration (also called Tier 1 in [3,14]) was a subset of the DXC10 and DXC11 configurations. More specifically, the DC load and one AC load were not part of DXC09.

4. Faults in electrical power systems

In this section we discuss faults that can occur in electrical power systems, and how these faults are examples of general fault types that ProDiagnose handles.

4.1. Electrical power system faults and their impact

EPSs can fault for many reasons, and here we briefly discuss some specific faults, how they were handled, and their ultimate impact. We focus on aircraft and spacecraft, since the ADAPT EPS is representative of their power systems.

4.1.1. Faults in aircraft

On September 15th, 2006, an EasyJet Airbus A319 suffered an in-flight EPS failure, which disabled the plane’s communication, instrumentation, and autopilot, among other systems. This forced the crew to fly the plane manually for the last part of the flight.6

The cause of this failure was an intermittent fault in the left electrical network of the EPS. A defect in the left generator control unit caused a current sensor onboard to incorrectly report a positive current flow. This resulted in tripping a circuit breaker, rendering the left electrical network inoperable. Furthermore, in a situation such as this, the crew must reconfigure the EPS manually by diverting power from a backup source, as the plane had no system to automate this process. The reconfiguration failed.

An important outcome of this incident was the realization that reconfiguration of the EPS should occur automatically in the event of an individual electrical network failure, and appropriate hardware upgrades were considered. However, a diagnostic system that correctly isolates the initial incorrect current sensor readings could then provide that information to a system that keeps the suspect electrical network online, thereby possibly eliminating a reconfiguration altogether.

4.1.2. Faults in spacecraft

An electrical power system is typically vital to the functioning of the spacecraft in which it is employed. A failure of this system often results in the unfortunate loss of the vehicle itself, as illustrated in the following three incidents.

On August 22nd, 1993, The NASA Mars Observer was lost after failing to re-establish communications shortly before entering Mars orbit. One of the potential causes is an EPS failure resulting from a regulated power bus short circuit.

On January 26th, 2000, the USAF FalconSAT-1 was launched, but the mission was terminated about a month later due to the inability of the satellite’s EPS to keep its batteries charged during daylight.

On November 2nd, 2006, the NASA Mars Global Surveyor ceased contact during its third extended mission. A faint signal received days later from the spacecraft indicated that it had entered safe mode, but it was subsequently lost. The failure was probably caused by an incorrect software update the previous year, which resulted in the spacecraft’s solar arrays being incorrectly driven to an immovable state. This caused an incorrect fault diagnosis of the gimbal motor, and subsequent action of the Global Surveyor to reposition itself based on this diagnosis resulted in the usable battery onboard to overheat.

4.2. Fault types

When defining fault types, we consider a system as consisting of parts.7 In the case study of this article, a system would be an electrical power system, and a part could be a battery, a wire, a voltage sensor, a temperature sensor, etc.

There are many dimensions along which systems fault. Diagnostic techniques and systems typically vary accordingly, since it is very difficult to develop approaches that are able to detect or isolate along all of these dimensions equally well and with equal computational efficiency. We now discuss the dimensions that we focus on in this article in more detail.

One dimension is speed of fault progression, where we distinguish between faults that progress very quickly (abrupt faults) versus faults that progress very slowly (incipient faults). Another dimension is fault dynamics, where one typically distinguishes between persistent and intermittent faults. A third dimension is fault cardinality, where it is fruitful to distinguish between continuous (or parametric) faults and discrete faults. As an example, ‘stuck high’ is a discrete fault, while ‘stuck at X’, where X is a parameter that can vary over a real-valued interval, is considered a continuous fault. A fourth dimension is fault dependency, namely whether faults are independent or dependent; common cause faults and cascading faults are clear examples of dependent faults. A fifth dimension is fault scope; models and algorithms may make the single fault assumption or not.

---

6 http://www.aaub.gov.uk/publications/formal_reports/4_2009_g_ezac.cfm.
7 A ‘part’ is either a ‘component’ or a ‘sensor’ according to the terminology used in this article. In the DXC10 documentation [43], the term ‘component’ rather than ‘part’ is used.
In this article, we focus on:

- Abrupt and incipient faults.
- Continuous and discrete faults.
- Only independent faults, in the sense that we do not consider common causes or cascading faults.
- We do not make the single-fault assumption.

4.2.1. Fault dynamics: Persistent versus intermittent faults

We first consider how a persistent fault takes place. Let $P(t)$ denote a measurable property of a part at time $t$, $P_n(t)$ denote the value of the property before fault occurrence, and $P_f(t)$ denote the value of the property after fault occurrence. More formally, let $t^*$ be the time of fault occurrence. We then have [43]:

$$
P(t) = \begin{cases} 
P_n(t): & t < t^* \\
P_f(t): & t \geq t^* \end{cases} \tag{1}$$

Consequently, the fault remains present at any time $t$ from $t^*$ onward, as Fig. 2 illustrates.

Intermittent faults differ from the persistent type in that the fault will not remain present continuously from $t^*$ onward. In other words, the fault will ‘disappear’ at some random time $t^{RD}$ after $t^*$, only to reappear again at some later random time $t^{RA}$. This cycle carries on indefinitely. Fig. 3 illustrates such intermittent behavior.

4.2.2. Fault progression: Abrupt versus drift faults

An abrupt fault is a sudden abnormal change in behavior, for example of a part in an electrical power system. This abnormal behavioral change may be an offset in a measurable property of the part. The persistent fault in Fig. 2 can be considered abrupt, as the slope of the measurements taken around $t^*$ is almost vertical. Abrupt faults are relative to the time-scale at which the diagnostic algorithm operates. Diagnosis of abrupt faults can occur quickly, especially if the behavioral change is large in magnitude.

A simple model for a persistent abrupt fault at time $t \geq t^*$ is defined as follows [43]:

$$
P_f(t) = P_n(t) + \Delta p, \tag{2}$$
where $\Delta p$ is an arbitrary positive or negative constant representing the offset magnitude [43]. The offset magnitude is shown visually as a voltage decrease around $t^*$ in Fig. 2. We do not, in general, know the values of $\Delta p$ or $t^*$ ahead of time, however once $\Delta p$ occurs at time $t^*$, it does not change (the fault is persistent).

We will focus on two types of persistent abrupt faults: abrupt offset faults and abrupt stuck faults. Fig. 2 can be classified as an abrupt offset fault, and we will revisit it now, in Fig. 4. In this figure, the added light green box, denoted as $R_n$, represents the nominal range in which this voltage sensor is considered healthy. As long as the sensor readings are within $R_n$, the sensor is deemed to be healthy. $R_n$ itself may depend on other evidence within the EPS, meaning that other changes within the EPS may change what $R_n$ is for this sensor at any time. In Fig. 4, the sensor’s reading (voltage) suddenly drops downwards out of $R_n$ at around $t^*$. Assume that the mode of the EPS, excluding this voltage sensor, has not changed. Thus, $R_n$ for the sensor remains unchanged from before $t^*$, and we conclude that the sensor itself is offset (faulty). This illustrates a basic abrupt offset fault; $\Delta p$ is large enough for the offset (voltage drop) to be quite visible.

A key challenge with abrupt faults is that $\Delta p$ can be very small, constituting small-magnitude offset faults. Fig. 5 illustrates an abrupt battery degradation of very small magnitude. This degradation, caused perhaps by a short in a load somewhere downstream from the battery, is shown as an abrupt voltage drop from a voltage sensor monitoring the battery. Sensor noise, while not reflected in (2), can mask the fault almost completely. In Fig. 5, the magnitude of sensor noise would make diagnosis very difficult without first filtering sensor readings. Another issue is the large number of states needed in a discrete random variable, if a naive approach is used by representing a large number of offset intervals directly. We would like to avoid such a large number of states.

A special case of a persistent abrupt fault arises when a sensor abruptly becomes stuck at a specific value. This case is defined as follows [43]:

$$P_f(t) = c_s,$$

(3)

---

8 The points at which the intermittent fault appears in Fig. 3 can also be considered abrupt, however intermittent faults are not persistent.

9 If the mode of the EPS had changed at around $t^*$ such that the voltage offset could be explained by that fact, then the nominal range for the sensor would have shifted downward to match this explanation.
where $c_s$ is a constant which represents the value the sensor is stuck at after $t^*$ [43]. Fig. 6 illustrates this special case. The noise associated with this sensor’s readings are shown before the point of fault occurrence, $t^*$. This sensor is moderately noisy, but can have short periods in which it repeatedly returns exactly the same reading (highlighted in Fig. 6 by red ovals, denoted as $O_1–O_5$). After $t^*$, all noise disappears from the sensor readings. This is not characteristic for this type of sensor, thus suggesting a fault.

Faults are not always abrupt, but may instead take some time to grow. The abnormal behavioral change of a part in an EPS due to a slowly growing fault, called an incipient or drift fault, usually is not noticed immediately after $t^*$. Rather, the fault becomes ‘visible’ when it reaches a certain magnitude. For example, a drifting sensor could output values that start gradually increasing at a roughly linear rate (Fig. 7a). A simple model for a persistent drift fault at time $t \geq t^*$ is defined as follows [43]:

$$P_f(t) = P_n(t) + m(t - t^*),$$

in which $m$ is the slope [43]. As seen in Fig. 7b, drift faults may not be so obvious at first, due to sensor and other noise not reflected in (4).

Sometimes different fault types result in very similar sensor readings, especially in the short term, and there can thus be ambiguity as to which fault(s) happened. Fig. 8 represents a fan failure. Power is applied throughout the time interval shown on the graph, but the fan suddenly stops working at $t^*$. As a result, both the current draw of the fan and the RPM of its blades drop to zero, as captured by the fan’s RPM sensor and a current sensor in series with the fan. Due to the inertia of the fan blades, they gradually stop spinning sometime after the fault occurrence, so the fault may appear to be a drift fault for a while. However, in this case, the fault is indeed abrupt, characterized by the immediate drop to zero of the current draw (Fig. 8).
5. Diagnostic Bayesian network model

A diagnostic Bayesian network (BN) model of a physical system, for example an EPS, consists of structures modeled after the physical components (parts) of the system \([3,14]\). In this section we discuss the BN node types used, and include in our discussion how these nodes are aggregated into network sub-structures that represent the parts of an EPS. Finally, we show how the BN model of a full EPS is constructed from these sub-structures.

5.1. Types of Bayesian network nodes

We partition the nodes in the network into non-evidence (output and hidden) and evidence (input, also called observed) nodes. Evidence nodes are always observed, meaning that at any given time, their state is known with 100% certainty. The process of assigning a specific state to an evidence node is referred to as clamping.

Non-evidence nodes are:

- Health nodes \(H\): A health node \(H \in H\) is used to represent the health status of a part. \(H_S\) are the health nodes for sensors, and \(H_C\) are the health nodes for components. A health state \(h \in \Omega(H)\) is selected based on the result of BEL or MPE computations, and reflects the health status of the part. We assume that \(\Omega(H)\) is partitioned into nominal (or normal) states \(\Omega_n(H)\) and faulty (or abnormal) states \(\Omega_f(H)\).
- Attribute nodes \(A\): An attribute node \(A \in A\) is used to represent a hidden state, such as voltage or current flow.

Evidence (observed) nodes are:

- Sensor nodes \(S\): A sensor node \(S \in S\) represents the present reading of a sensor \(s\). This reading is a discretized real or boolean value, which represents, respectively, a range of states for real-valued sensors or the state of 0 or 1 for a boolean sensor. A corresponding sensor reading \(s(t)\) is clamped as evidence in the network (its state at time \(t\) is known with 100% certainty). Sensor nodes, further discussed in Section 6.3, belong to the set of evidence nodes \(S \subseteq E\).
- Command nodes \(C\): A command node \(C \in C\) represents a command issued to a part. An example would be a command to open or close a relay. Command nodes belong to the set of evidence nodes \(C \subseteq E\).
- Stuck nodes \(ST\): A stuck node \(ST \in ST\) is used to clamp evidence for, or against, a stuck fault (see Fig. 6). Stuck nodes, further discussed in Section 6.4, belong to the set of evidence nodes \(ST \subseteq E\).
- Intermittent nodes \(I\): An intermittent node \(I \in I\) is used to clamp evidence for, or against, an intermittent fault (see Fig. 3). Intermittent nodes, further discussed in Section 6.5, belong to the set of evidence nodes \(I \subseteq E\).
- Drift nodes \(DR\): A drift node \(DR \in DR\) is used to clamp evidence for, or against, a drift fault (see Fig. 7). Drift nodes, further discussed in Section 6.5, belong to the set of evidence nodes \(DR \subseteq E\).
- Delta nodes \(D\): A delta node \(D \in D\) represents the difference (delta) between the current reading \(s(t)\) of a sensor \(s\) and its previous reading \(s(t-1)\). They are useful for detecting short-term behavior, as shown in Fig. 8. Delta nodes, further discussed in Section 6.3, belong to the set of evidence nodes \(D \subseteq E\).
5.2. Bayesian network structures

The parts of an electrical power system (EPS) are represented by specific structures within a Bayesian network (BN). Here we partition the parts into two categories: sensors and components. Any part which is not a sensor can be considered a component. Sensors tend to have the same overall BN structure, whereas component structures may vary, depending on the part.

Fig. 9a shows the structure of a boolean sensor. This structure consists of an S node and an $H_S$ node. Potentially discretized readings (see Section 6.2) of the sensor are clamped to the S node, and the corresponding health of the sensor, given this evidence and other evidence from the BN, is provided by the $H_S$ node. The S node is a child of the $H_S$ node, thus the evidence clamped to the S node will directly influence the posterior state of the $H_S$ node. In a complete EPS BN, many factors will influence this posterior state.

Fig. 9b expands on the basic boolean sensor structure to better capture behavior associated with real-valued sensors, such as voltage or current sensors. One distinguishing factor between the structure of a real-valued sensor and that of a boolean sensor (Fig. 9a) is the presence of other nodes, besides the S node, that are children of the $H_S$ node for real-valued sensors. This type of sensor typically has many faulty states that require additional evidence nodes, such as ST, I and DR nodes. These nodes are optional, and it is possible for a real-valued sensor's structure to be as shown in Fig. 9a.

Fig. 10a shows a sample BN representation of a component with no sensor associated with it. The C (command) node will be present for components that are controlled by an outside agent using commands, for example a relay that can be
opened or closed. For all other sensor-less components, no nodes exist in the structure in which evidence is clamped. For such components, the posterior health state of the $H_C$ node will be influenced by evidence provided outside this structure (other sensors within the EPS, for example). The $A$ node represents the internal state of the component, and usually is a reflection of the health state.

Fig. 10b takes the sensor structure from Fig. 9a and attaches it to the basic component structure to form a combined component and sensor BN structure. This structure is used for a component that has a boolean sensor directly monitoring it.\(^{10}\) Notice in Fig. 10b that the sensor structure’s $S$ node is directly connected to an $A$ node, which represents the internal state of the component. Thus, the evidence clamped to the $S$ node will have a strong influence over the health state of the component. The rest of the BN will also exert influence, as it too is connected directly to the same $A$ node. The evidence clamped to the $S$ node combined with the evidence clamped to other nodes in the Bayesian network will help differentiate between many faults in the component and sensor arrangement of a part, most notably component failures versus sensor faults.

5.3. Constructing the EPS model

We now focus on how the BN structures discussed in Section 5.2 interconnect to form a complete BN representation of an EPS \cite{14}. Fig. 11 shows the complete BN for the ADAPT EPS (DP2); the yellow boxes highlight some of the structures discussed in Section 5.2.

\(^{10}\) Physical components with attached real-valued sensors are not modeled in a uniform way in the Bayesian network, and thus we save the discussion of these components for Section 5.3.
Table 2
Statistics on the ADAPT DP2 and DP1 BNs. Average node cardinality, for any BN, refers to the average number of parents of the nodes. For DP2 and DP1 average node cardinality is equal to two decimal places. Dividing edge count by node count (fourth column) shows the sparsity of the BNs, which makes them suitable for exact inference methods like arithmetic circuits. Treewidth gives insight to an upperbound for graph density, and was computed using the QuickBB approach developed by Gogate and Dechter [44]. The DP2 BN (Fig. 11) is mostly sparse except for some areas in and around the loads on the right side, thus, the treewidth value would have originated from this area.

<table>
<thead>
<tr>
<th>BN</th>
<th>Node count</th>
<th>Edge count</th>
<th>Edge count/Node count</th>
<th>Average node cardinality</th>
<th>Treewidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP2</td>
<td>494</td>
<td>602</td>
<td>1.22</td>
<td>2.84</td>
<td>4</td>
</tr>
<tr>
<td>DP1</td>
<td>172</td>
<td>224</td>
<td>1.30</td>
<td>2.84</td>
<td>6</td>
</tr>
</tbody>
</table>

Fig. 12. Connection of a real-valued sensor structure to a component structure. The blue A nodes (A₁ and A₂) are used as connection points for the rest of the BN. The D node is used to provide short-term behavioral characteristics (such as if fan blades are accelerating, decelerating, or neither) of the component in this model. The BN structure in the dotted box is a sensor structure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Statistics on the DP2 and DP1 BNs are given in Table 2. The modeling process and individual BN structures are similar in both DP2 and DP1 BN models.

Fig. 12 illustrates the BN representation of a component with attached real-valued sensor. The sensor structure (within the dotted box) is not directly attached to the component structure, but rather is indirectly attached using some A nodes. Node A₁ typically represents voltage, while node A₂ represents a sink. Sink A nodes provide evidence to the status of the wires which connect the sensors and components in an EPS. This status is usually the presence of current flow through a wire.

Node A₃ in Fig. 12 represents some other property of the component, which is usually current flow. Representing current flow makes sense given that node A₃ is a child of node A₂, a sink node. Node A₄ represents the internal state of the sensor structure.

To construct the EPS model, part structures are connected to one another either directly or in the manner shown in Fig. 12. In the BN shown in Fig. 11, the battery temperature and circuit breaker boolean sensor structures are directly connected to each other, whereas the current flow sensor is connected to a node of type A₃. The latter follows the same general structure shown in Fig. 12: sensor structures are connected to a node of type A₃, while component structures are connected to nodes of type A₁ and A₂.

To provide some insight into the latter part structure connection method, consider sink A nodes (Figs. 11 and 12). Sink nodes connect to each other to form chains, with each chain representing one circuit in the EPS. Each sink node has at most one parent and one child that are also sink nodes. As sink nodes typically represent the presence of current flow through a wire, they provide valuable additional evidence to part structures. For example, if a circuit breaker is tripped, that should prevent current flow through the wire connected through the circuit breaker. If a current flow sensor on the same circuit is still measuring current flowing through the wire, then either the circuit breaker is not actually tripped, or the current flow sensor is faulty.

6. Diagnostic algorithm

Our ProDiagnose diagnostic algorithm (see Fig. 13) performs processing and inference on a data stream provided by an environment [3,14]. Environments could be a physical system, such as an electrical power system, or a simulator. Environments give to ProDiagnose two different types of data: sensor readings, s(t), and commands, c(t) (Fig. 13). Sensor readings, s(t), are provided to ProDiagnose in regular intervals, called the global sample rate. The global sample rate is typically determined by the rate at which sensors produce readings, and the fastest sample rate of all the sensors in the environment
is taken to be the global sample rate.\textsuperscript{11} If the global sample rate is 5 Hz, then $s(t)$ is processed every 0.2 seconds. The environment is synchronized with ProDiagnose according to the global sample rate.

Commands, $c(t)$, are provided to ProDiagnose whenever they are issued by the environment (Fig. 13). An example of $c(t)$ would be a command to close a relay in an EPS. ProDiagnose processes $c(t)$ according to the global sample rate. This requires ProDiagnose to maintain a queue for any $c(t)$ provided between sensor processing intervals. For example, if the sample rate is 5 Hz, and we start ProDiagnose at $t = 0$ seconds, then any $c(t)$ received in the interval (0.2, 0.4) seconds is queued and processed at $t = 0.4$ seconds.

Processing of $s(t)$ and $c(t)$ takes three stages: pre-processing, inference, and post-processing. Pre-processing involves translating $s(t)$ and $c(t)$ into evidence, $e(t)$. Pre-processing is discussed in greater detail in Sections 6.3 to 6.5 which discuss fault handling.

Inference computes answers to queries, $q(t)$, provided by ProDiagnose, given $e(t)$. The computational engine is the Arithmetic Circuit Evaluator (ACE), discussed in greater detail in Section 6.1. The output is in the form of a state set, $s(t)$, which contains a state selection for each node $X \in X$ in the BN.

During post-processing, $s(t)$ is processed and output as a diagnosis, $d(t)$, to an evaluator (Fig. 13). An evaluator could be a system that performs actions on an EPS given its health. An evaluator could also be the component of a simulator that scores $d(t)$, computing accuracy or other metrics found in Section 7.1.

The diagnosis at time $t$, or $d(t)$, where $d(t) \subseteq s(t)$, contains a state selection for each $H \in H$. ProDiagnose further filters $d(t)$ before sending it to the evaluator [3,14]. These filters employ adjustable parameters (currently set manually) used for suppressing specific diagnoses under certain criteria, such as if a new fault is diagnosed, or command issued. These parameters are given in units called cycles, and to convert cycles to time, simply divide the cycles by the global sample rate.

The important adjustable parameters are as follows:

\begin{itemize}
  \item diagnosisDelay, $c_{DD}$: Cycles to suppress all diagnoses. This parameter suppresses all diagnoses until after the specified period of time has passed from the beginning of ProDiagnose execution. diagnosisDelay is useful for filtering out nominal behavior that is unique to the startup phase of a system being monitored by ProDiagnose.
  \item commandOffset, $c_{CO}$: Cycles to suppress all diagnoses after a command issued. This parameter is useful for filtering out transients often caused by commands, such as closing relays in an electrical power system.
  \item faultDelay, $c_{FD}$: Cycles to suppress all diagnoses when an abnormal health state is observed in $d(t)$ (a fault), given that no fault was observed during the previous cycle. This parameter helps to filter out false positives at the cost of longer diagnosis times.
\end{itemize}

6.1. ACE

ProDiagnose’s inference engine (ACE) performs inference on arithmetic circuits, compiled from Bayesian networks. It supports two inference methods: sum-of-product and max-of-product [10–12]. The sum-of-product (SOP) method is based on computing the probability of evidence, and is used for $BEL(X, e)$ queries. A SOP query to the AC itself does not give the needed information required for ProDiagnose to update the health states of the $H$ nodes. Immediately following the SOP query, ACE differentiates the AC, and returns the posterior distributions for all variables in the Bayesian network. For each $H$ node, the health state with highest probability is taken from its posterior distribution and assigned as the new health state.\textsuperscript{12}

\footnote{\textsuperscript{11} We are making the simplifying assumption that the sample rate $s(t)$ of any sensor is a multiple of the global sample rate.}

\footnote{\textsuperscript{12} In the case of a tie (multiple states having the highest probability), the first state processed is taken to be the state with highest probability.}
The max-of-product (MOP) method computes MPE(\(\mathbf{e}\)) and its probability. For each \(H\) node, its MPE state will correspond to its new health state. ProDiagnose will update the health state of each \(H\) node based on its MPE computed by the MOP query.\(^{13}\)

Note that taking the highest probability state as the new state, after SOP inference, is not the same as MPE computed by MOP inference. While ProDiagnose supports both SOP and MOP inference, and we have experimented with both, MOP is currently preferred due to faster inference times compared to SOP (see Section 7.3). Unlike for SOP, differentiation is not required for MOP queries to return the states of the \(H\) nodes to ProDiagnose.

### 6.2. Discretization

Before describing the fault handling techniques of ProDiagnose, it is important to discuss how discretization is accomplished. In some of our BN nodes, a discrete value, or state, represents a continuous range of values, or an interval. A threshold is a boundary between intervals.

Formally, a real-valued sensor contains values which form a contiguous subset of the real line \(\mathbb{R}\). We take this subset and partition it into continuous intervals. For example, if a real-valued sensor contained the entire line \(\mathbb{R}\), then we would partition like so:

\[
(-\infty, t_1](t_1, t_2](t_2, t_3] \ldots (t_{n-1}, t_n](t_n, \infty),
\]

where \(t_i\) represents the upper threshold for interval \(i\). In this article, we refer to the upper threshold of an interval as simply the threshold. Looking at (5), the threshold for interval \((t_1, t_2]\) (interval 2) would be \(t_2\).

### 6.3. Handling offset faults

Abrupt offset faults (see Section 4.2.2) are handled in a few different ways, depending on the behavior of the offset fault. The following examples are for abrupt offset faults, though many non-abrupt offset faults can be diagnosed using these same techniques.

Offset faults of a non-small magnitude (Fig. 4) are handled using simple clamping of evidence, usually to a sensor’s \(S\) node. For example, suppose that at time \(t^*\) a voltage sensor starts giving readings that are about 10 volts lower than the actual voltage. If we assume that a 10 volt drop is enough to cause the discretized value which we clamp to the sensor’s \(S\) node to change, then the state of the \(S\) node will change respectively. Assuming all other evidence to the BN remained the same, this could suggest the sensor is offset, and this may be reflected in a corresponding \(H\) node state of ‘offset’.\(^{14}\)

Certain sensor structures also require use of the \(D\) node (see Section 5.1). \(D\) nodes provide additional evidence in terms of dynamic behavior. Consider Fig. 8. Without the use of \(D\) nodes to provide evidence about the fan blades’ angular momentum, the fault could be misdiagnosed as a sensor fault with the fan’s RPM sensor or current flow sensor [14].

Now we discuss how to handle offset faults of a small magnitude. Referring to the same example, suppose that the offset was small enough such that the state of the \(S\) node remained the same. Assuming the rest of the evidence provided to the BN remained the same, this fault would go undiagnosed.

A naïve approach to address this problem would be to simply increase the state space cardinality \(|\Omega(S)|\) of the \(S\) node, thereby allowing smaller offset faults to be diagnosed. However this poses two issues: more states added to these nodes increases BN complexity and hence computational time, and more states for the \(S\) nodes increase the risk of false positives due to noise.

We instead take another approach and increase the amount of evidence provided to the Bayesian network. This is accomplished by adding a new node type, called a change \(CH\) node (defined in Section 5.1), and incorporating supplemental processing (discussed in Section 6.3.2) in the diagnostic algorithm to calculate the additional evidence clamped to the \(CH\) nodes [15]. In an ADAPT Bayesian network, \(CH\) nodes on average contain 2–3 states (\(2 \leq |\Omega(CH)| \leq 3\) on average). The number of these nodes is a fraction of the number of \(S\) nodes, which helps to minimize the total size of the joint distribution represented by the BN. The supplemental processing filters out sensor noise while generating the \(CH\) node evidence, which helps to minimize false positives due to noise.

#### 6.3.1. Offset fault handling in the Bayesian network

We use \(CH\) nodes to solve the problem of diagnosing offset faults of a small magnitude. \(CH\) nodes are incorporated into the Bayesian network for two main purposes: To increase the evidence provided to a single part structure, and to increase the amount of evidence available to several part structures. First we discuss the single part case, then a case involving multiple parts in a load bank.

---

\(^{13}\) In the case of more than one MPE, the last MPE returned is used.

\(^{14}\) If other evidence provided to the BN also changed, an alternate explanation could be a voltage drop within the system. If this is indeed the case, the voltage sensor’s \(H\) node would most likely remain at a state of ‘healthy’. It is also possible for an \(S\) node’s state to not change, but other evidence provided to the BN changes to suggest an offset fault.
Connection of part structures for a battery, a circuit breaker, and a voltage sensor to a \( CH \) node for the purpose of diagnosing offset faults that are of a small magnitude. The connection \( A \) nodes (marked as \( AC \)) form a chain that help propagate voltage-related evidence upstream from the voltage sensor, and are used to connect the depicted part structures to the rest of the BN.

### Table 3
Conditional probability table (CPT) of the \( CH \) node from Fig. 14 (with label ‘Change_voltage_e140’ in the ADAPT BN). Each row shows the probabilities for the \( CH \) node's nominal and drop states (first two columns) for each combination of states from the battery structure's \( A \) node, circuit breaker structure's \( A \) node, and voltage sensor structure's \( H \) node (columns 3–5). Since the probabilities remain uniform for all rows when the \( H \) node is unhealthy, we simplified the table by combining all these unhealthy states into (not healthy).

<table>
<thead>
<tr>
<th>( CH )</th>
<th>( A ) (battery)</th>
<th>( A ) (breaker)</th>
<th>( H ) (sensor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal</td>
<td>Drop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.95</td>
<td>0.05</td>
<td>enabledHigh</td>
<td>closed</td>
</tr>
<tr>
<td>0.05</td>
<td>0.95</td>
<td>enabledLow</td>
<td>healthy</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>disabled</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>enabledHigh</td>
<td>open</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>enabledLow</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>disabled</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>enabledHigh</td>
<td>closed</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>enabledLow</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>disabled</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>enabledHigh</td>
<td>open</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>enabledLow</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>disabled</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 14 shows part structures shown highlighted in the bottom left corner of Fig. 11. The \( CH \) node in Fig. 14 is present in the top-most battery structure shown in Fig. 11.\(^{15}\) This \( CH \) node is used to provide extra evidence regarding the battery structure’s long-term behavior. This information is used to diagnose the abrupt battery degradation fault depicted in Fig. 5. The evidence provided by the \( CH \) node is derived from some sensor’s readings within the EPS, which we call the source sensor. In Fig. 14, the source sensor is the voltage sensor. The battery is referred to as the target, as it is the battery’s long-term behavior we are capturing through the \( CH \) node, and hence why we use the closest voltage sensor to the battery as the source sensor.

If the source sensor is faulty and producing incorrect readings, then would the evidence provided by the \( CH \) node also be incorrect? The answer is yes, and we do not want false evidence incorrectly influencing other part structures in the BN model. For this reason, the \( CH \) node is also connected directly to the \( H \) node of its source sensor structure. The evidence clamped to the \( CH \) node will only have influence if the source sensor is deemed to be healthy. The \( CH \) node’s conditional probability table, given in Table 3, shows how this is accomplished.

---

\(^{15}\) \( CH \) nodes were not used for the other two battery structures due to a smaller state set \( \Omega(H) \) being used for these two structures that did not require the extra node.
Fig. 15. Bayesian network representation of a load bank. This bank consists of four loads, but only one sensor, a current flow sensor upstream of the load bank. The load property $A$ nodes (maroon – marked as $A_P$) are used to propagate additional evidence provided by the $CH$ node to the load structures. This is accomplished through the tree-like structure (white $A$ nodes – marked as $A_1$–$A_6$).

Fig. 16. Subset of the tree-like structure from Fig. 15, showing sample CPTs for three $A$ nodes. The states represent power, and each node CPT sums the power of its parents. We assume that nodes $A_1$ and $A_2$ only have one parent each (see Fig. 15).

Connecting the $CH$ node to its source sensor structure’s $H$ node solves the problem of faulty evidence clamped to the $CH$ node, but what about the circuit breaker depicted in Fig. 14? The edge from the circuit breaker structure’s internal state $A$ node to the $CH$ node is used for much the same reason as the connection to the source sensor structure’s $H$ node. Any part located along the path from source sensor to target, that could change the behavioral characteristics of the property monitored by the source sensor, needs to be connected to the $CH$ node. This connection will always be done using the part structure’s internal state $A$ node.

While $CH$ nodes can be used to provide extra evidence to one target part structure, they also can provide extra evidence to multiple target part structures. Fig. 15 illustrates this usage in a load bank, which is simply multiple loads connected by a common power source through a circuit breaker or relay. In general, it is not uncommon to have a bank with many loads, but very few sensors to provide evidence concerning their statuses. Some of these loads may have no sensors at all, and hence diagnosing these loads becomes difficult. We now discuss how to solve this problem.

The $A$ nodes that connect directly to the load structures in the bank in Fig. 15 (marked as $A_P$) are not internal state nodes, but rather are representative of the property that the $CH$ node’s source sensor measures. Let us call these $A$ nodes load property nodes. Load property nodes are connected to the $CH$ node by a tree-like structure, represented in Fig. 15 by $A$ nodes $A_1$–$A_6$. This tree-like structure serves to simplify the complexity of the $CH$ node’s CPT by distributing it among the nodes in the structure [45]. In Fig. 15, the source sensor of the $CH$ node is the current flow sensor. Therefore, the $CH$ node

---

16 This is another reason to pick a source sensor that is close to the target, as the more parents the $CH$ node has, the larger its CPT.
Table 4
Conditional probability table (CPT) of the $CH$ node from Fig. 15 (with label ‘Change_current_it167’ in the ADAPT BN). This table is laid out in the same format as Table 3. Some simplifications were made to the CPT so it would fit, including taking out some intermediary states (represented by ‘...’ from the $CH$ node and its parents.

<table>
<thead>
<tr>
<th>CH</th>
<th>w0</th>
<th>w30</th>
<th>w60</th>
<th>...</th>
<th>w420</th>
<th>w450</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.05</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
<td>0.0</td>
<td>w0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.05</td>
<td>0.0</td>
<td>...</td>
<td>0.0</td>
<td>0.0</td>
<td>w60</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.05</td>
<td>...</td>
<td>0.0</td>
<td>0.0</td>
<td>w90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>w0</td>
<td>w0</td>
<td></td>
</tr>
<tr>
<td>w30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>w60</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4 gives the CPT for another $CH$ node. This $CH$ node, like the $CH$ node depicted in Fig. 15, provides additional evidence to a load bank, but in a much more complex environment. Looking at Table 4, it is apparent that each $AP$ node has more states to begin with, and possibly more loads in the bank. Especially in the latter case, the tree-like structure could incorporate more
nodes to further reduce the CPT size. Referring to the properties of a binary tree, the tree-like structure will only need to be \( \log(|A_P|) \) deep, where \(|A_P|\) is the number of \( A_P \) nodes.

6.3.2. Offset fault handling in the diagnostic algorithm

We have already covered offset fault handling in regard to \( S \) nodes, so in this section we focus on offset fault handling in regard to \( CH \) nodes. Evidence is generated for \( CH \) nodes by a process called cumulative sum (CUSUM) [14]. CUSUM is a sequential analysis technique used in statistical quality control [16]. CUSUMs can be used for monitoring changes in a continuous process’ samples, such as sensor readings, over time. Let \( \delta_p(t) \) represent the CUSUM of sensor \( p \) at time \( t \). Then, taking \( \bar{s}_p(t) \) to be the unweighted reading from sensor \( p \) at time \( t \), we formally define CUSUM as [16]:

\[
\delta_p(t) = [s_p(t) - s_p(t - 1)] + \delta_p(t - 1).
\]

To help filter out sensor noise, a weighted average over time of the raw sensor readings is taken, which form the basis for CUSUM computation [15]. Let \( \bar{s}_p(t) \) be the weighted average of readings \( \{s_p(t - n), \ldots, s_p(t)\} \) for sensor \( p \) at time \( t \). Specifically, we have:

\[
\bar{s}_p(t) = \sum_{i=0}^{n} s_p(t - i) w(t - i),
\]

where \( s_p(t - i) \) is the raw sensor reading and \( w(t - i) \) is the weight at time \( t - i \). The summation in (7) is over all sensor readings from time \( t \) to time \( t - n \). In other words, we keep a history of the past \( n \) sensor readings. The values of \( n \) and of all weights \( \{w(t - n), \ldots, w(t)\} \) are configurable and set based on experimentation.

The weighted sensor averages in (7) are used by ProDiagnose when calculating CUSUM [14,15], and thus (6) is modified accordingly:

\[
\tilde{\delta}_p(t) = [\bar{s}_p(t) - \bar{s}_p(t - 1)] + \delta_p(t - 1),
\]

where \( \tilde{\delta}_p(t) \) is the weighted average CUSUM of sensor \( p \) at time \( t \). Weighted averages help to smooth out spikes and other noise in sensor readings that could otherwise lead to false positives or false negatives. The \( \tilde{\delta}_p(t) \) values are then discretized and clamped to \( CH \) nodes as evidence.

Using CUSUM has two main benefits related to our hybrid diagnosis approach: It amplifies offset faults of small magnitude (see Section 4.2.2) and normalizes by shifting offsets to a normalized \( y \)-value, such that a single set of thresholds can apply to many different continuous offsets [15]. An example of the latter would be two voltage sensors with different nominal voltages, say 4 V and 22 V. CUSUM would normalize these nominal voltages to zero. See Fig. 17 for an illustration of these two benefits.

For any sensor \( p \), before CUSUM can start, \( \bar{s}_p(0) \) needs to be computed. This computation, which we call calibration, begins when \( p \) starts producing readings.\(^{20}\) CUSUM starts once \( \bar{s}_p(0) \) is computed.

---

\(^{20}\) This initial weighted sensor average doesn’t necessarily use the same \( |n| \) (see (7)) as the subsequent weighted averages. For the initial weighted average, \( |n| \) is usually much larger.
6.4. Handling stuck faults

Recall the behavioral pattern of a stuck fault from Fig. 6. Stuck faults are handled by using sensor readings accumulated over time, looking for subsequent readings that are close enough to be considered equal [14]. The result of this process is then added to the BN as evidence.

6.4.1. Stuck fault handling in the Bayesian network

Stuck fault handling utilizes one node type, ST, in the BN. Recall from Fig. 9b that ST nodes serve as children of H nodes in sensor structures. Positive evidence clamped to an ST node suggests with very high probability the presence of a stuck fault. For any given sensor structure, the sensor reading used for evidence with the ST node is also used to derive evidence for the ST node.

Referring to Table 5, ST nodes have three states: negDelta and posDelta correspond to groups of subsequent sensor readings that are different from one another, and zeroDelta corresponds to groups of subsequent sensor readings that are close. If the H node is not stuck, there is still a significant probability of subsequent sensor readings being close, due to external discretization of sensor readings.

6.4.2. Stuck fault handling in the diagnostic algorithm

Stuck faults are calculated by keeping track of close subsequent readings by use of a counter. When this counter reaches a certain value, evidence is clamped to ST = zeroDelta to suggest a stuck sensor. We determine if two sensor readings are close by taking their difference and comparing with a configurable epsilon value, \( \epsilon \). More formally, two sensor readings \( s_p(t) \) and \( s_p(t - 1) \) are considered close if the following is true:

\[
|\Delta s_p(t)| < \epsilon,
\]

where \( \Delta s_p(t) = s_p(t) - s_p(t - 1) \). The counter resets when two subsequent sensor readings are not considered close. Note that if \( ST = zeroDelta \) and the counter resets for that ST node, the state of that node will also change to either negDelta (if \( \Delta s_p(t) \leq -\epsilon \)) or posDelta (if \( \Delta s_p(t) \geq \epsilon \)). The posterior probability of the corresponding sensor actually being stuck thus decreases, assuming that all other evidence stays the same.

6.5. Handling intermittent and drift faults

We now discuss how to solve the problems of diagnosing intermittent and drift faults. Section 6.5.1 discusses intermittent and drift fault handling in the BN, Section 6.5.2 discusses intermittent fault handling in the diagnostic algorithm, and Section 6.5.3 discusses drift fault handling in the diagnostic algorithm.

6.5.1. Intermittent and drift handling in the Bayesian network

Intermittent and drift fault handling utilize a single node type each in the BN: the I and DR nodes, respectively [5,15]. Referring to Fig. 9b, we observe a DR \( \leftarrow \) H \( \rightarrow \) I pattern. This allows for the evidence provided by the I and DR nodes to strongly influence the posterior state of the H node. The evidence clamped to these nodes are derived from sensor readings. For a specific sensor structure in the BN, the evidence provided by the I, DR, ST and S nodes are all derived from the same sensor in the EPS.

Table 6 presents the CPTs for both the I and DR nodes. These CPT structures are similar to that of the ST node’s CPT (Table 5), with two notable differences. The cardinality of the H node has increased, to support the extra fault types compatible with intermittent and drift faults, and the cardinality of the I and DR nodes have decreased. Looking at Table 6, we have two extra parent states: intermittentOffset and drift. These two states are present for intermittent faults and drift faults, respectively. The I and DR nodes themselves only have two states: nominal and faulty.

The reason for having only two states for the I and DR nodes ties into the other notable difference in their CPTs. Unlike with the ST node’s CPT, the probabilities here are either 1 or 0. This implies that when the diagnostic algorithm assumes

---

Table 5
Conditional probability table (CPT) of the ST node from Fig. 14 (with label ‘Stuck_voltage_e140’ in the ADAPT EPS Bayesian network). Each row shows the probabilities for the ST node’s states (first three columns) for each state from the voltage sensor’s H node (fourth column). When the ST node is clamped as zeroDelta, the probability that the H node’s state will be stuck increases greatly in probability.

<table>
<thead>
<tr>
<th>ST</th>
<th>negDelta</th>
<th>zeroDelta</th>
<th>posDelta</th>
<th>H (Sensor)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.4999</td>
<td>0.0002</td>
<td>0.4999</td>
<td>healthy</td>
</tr>
<tr>
<td></td>
<td>0.4999</td>
<td>0.0002</td>
<td>0.4999</td>
<td>offsetToLo</td>
</tr>
<tr>
<td></td>
<td>0.4999</td>
<td>0.0002</td>
<td>0.4999</td>
<td>offsetToHi</td>
</tr>
<tr>
<td></td>
<td>0.0001</td>
<td>0.9998</td>
<td>0.0001</td>
<td>stuck</td>
</tr>
</tbody>
</table>
Table 6
Conditional probability table (CPT) of the I and DR nodes depicted in Fig. 14 (with labels ‘Intermittent_voltage_e240’ and ‘Drift_voltage_e240’ in the ADAPT BN). Each row shows the probabilities for both the I node’s states (first two columns) and the DR node’s states (last two columns), given the states of the voltage sensor’s H node (third column).

<table>
<thead>
<tr>
<th>I (sensor)</th>
<th>H</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal</td>
<td>Faulty</td>
<td>Nominal</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>healthy</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>offsetToLo</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>offsetToHi</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>offsetToMax</td>
</tr>
<tr>
<td>0.0</td>
<td>1.0</td>
<td>intermittentOffset</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>drift</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>stuck</td>
</tr>
</tbody>
</table>

Fig. 18. The graph from Fig. 3 with sample parameters overlaid to visualize the intermittent tracker. Parameter $r_{in} = [\text{min}, \text{max}]$ (nominal behavior observation range) is represented by the ranges bounded by the blue dashed lines below the graph, while $r_{if} = [\text{min}, \text{max}]$ (faulty behavior observation range) is represented by the ranges bounded by the red dotted lines above the graph. In this example, we assume the intermittent tracker starts tracking from the first fault occurrence ($t^*$), in which $r_{if}$ is marked as (1). The range specified within the red dotted lines, which themselves represent the min and max values of $r_{if}$ respectively, defines the range in which the tracker actively tracks any possible change of behavior from a faulty observation to a nominal one. The second fault occurrence where the intermittent tracker starts tracking is marked as (2). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.5.2. Intermittent fault handling in the diagnostic algorithm

Intermittent faults are calculated by using a tracking algorithm [5]. This algorithm works by tracking specific behavioral patterns associated with intermittent faults, as shown in Fig. 3. These patterns tend to follow a square wave. If this behavior persists long enough, then the algorithm deems an intermittent fault to have occurred, and clamps the offending I node to faulty.

What happens if we have a situation in which both the I and DR nodes are clamped to faulty for the same sensor structure (such as in Fig. 14)? Clearly, this is invalid. If we look at the CPTs for both the I and DR nodes, these two clamped states would imply that the H node’s state would be both intermittentOffset and drift with 100% certainty. This is impossible as the sum of all state probabilities must equal 1. Therefore the DA is configured as to never allow both the I and DR nodes for the same sensor structure to be clamped to faulty at the same time.
Drift faults are also handled by means of a tracking algorithm [15], which works by tracking specific behavioral patterns associated with drift faults, as shown in Fig. 7. The actual tracking is performed using CUSUM rather than raw sensor readings (see Section 6.3.2), since CUSUM helps amplify drift behavioral patterns. Drift faults tend to follow a linear incline or decline, and the tracker deems a drift fault to possibly have occurred if this behavior persists for a given period of time, at which the state faulty is clamped to the offending DR node.

The drift tracking algorithm is configured initially with parameters to specify the number of thresholds required to reach the range of time required before reaching each one [15]. Thresholds are denoted as constraint that if $i < j$, then $h_i$ must be reached by the tracker before $h_j$ for drift tracking to continue. The range of time required is a relative time interval, denoted by $r_h$, relative to the time at which $h_{i-1}$ was reached. $r_h$ is represented as a pair, $[\text{min}, \text{max}]$, where $\text{min}$ is the minimum amount of time, measured in cycles (see Section 6) required for threshold $h_i$ to be reached, and $\text{max}$ is the maximum amount of time, measured in cycles, that $h_i$ can be reached before the drift tracker rejects the observation and resets.

Fig. 19. The graph from Fig. 7 (a) with sample thresholds overlaid to visualize the drift tracker. This example has 4 thresholds, $h_1 - h_4$. The range of time to reach a threshold is given by $r_2$ for $h_2$, $r_3$ for $h_3$ and $r_4$ for $h_4$. The tracker starts tracking once threshold $h_1$ is reached (marked with (start) at the point where the graph crosses the $h_1$). At this point, threshold $h_2$ must be reached within the range of time specified by parameter $r_2$, or the observation is rejected by the tracker, and the tracker resets. Otherwise, the tracker continues tracking, and if $h_4$ is reached (as per this example), then the tracker nominates the observations as a drift fault. Tracking is using the current sensor’s CUSUM and not its raw values.
We can visualize the thresholds here as being consecutive ‘steps’ in terms of offset that the sensor readings in question must reach, constrained to a certain range of time, for the tracking to continue. Other parameters specify the value of each threshold, which allow for other patterns (besides linear) to be observed. The time intervals between thresholds define the slope of the line or curve defined by the thresholds. Fig. 19 gives a visualization of the tracker using the graph depicted in Fig. 7(a). Threshold \( h_1 \), being the first threshold, starts the drift tracker upon being reached. If \( r_2 = [2, 3] \) cycles, this means that threshold \( h_2 \) must be reached within a minimum of 2 cycles but no more than 3 cycles from when the drift tracker was started. If \( r_3 = [3, 6] \) cycles, then \( h_3 \) must be reached within 3 cycles from the time when \( h_2 \) was reached, but no more than 6 cycles from when \( h_2 \) was reached. In this example, \( h_4 \) has to be reached for a drift fault to be nominated by the tracker.

Note that for any \( r_{hi} \), \( min \) and \( max \) can be set independently of any other \( r_{hi} \). Referring to Fig. 19, \( r_2 \), \( r_3 \) and \( r_4 \) can all be set to unique \( min \) and \( max \) values. These ranges of time are important to distinguish a drift fault from an abrupt offset fault that may just happen to have an offset great enough to reach all drift-based thresholds at once (this is why \( min \) is important).

To prevent having another set of thresholds for negative (or declining) drift faults, the absolute value of the sensor readings in question are used, so the resulting drift, in terms of how the tracker sees it, is always positive (or inclining).

If all thresholds are reached within the time intervals given, the tracker nominates the observations as a drift fault, and the state for the DR node associated with the offending part’s BN structure is set to faulty. Once this happens the state is clamped indefinitely. The drift tracker algorithm works on any part that incorporates a DR node in its BN structure, and the tracker stays dormant until a possible drift fault arises, resetting back to the dormant state if the drift fault behavior disappears.

7. Experiments

In this section we report on experiments with the ADAPT EPS (see Section 3). Expanding upon previous work [3,5,14,15], we include results on inference times and methods, and aging of the ADAPT electrical power system. We also expanded the number of datasets used for experimentation; we have no previous results for the DXC11 datasets.

7.1. Data and methods

All experimentation was performed using scenario data obtained from the ADAPT EPS (see Fig. 1, Section 3), more specifically the DXC10 and DXC11 datasets. The BN model of the ADAPT EPS was introduced in Section 5.3 (see Fig. 11).

Table 7 gives an overview of the datasets used in the DX competitions (DXC10 and DXC11). The DP1 datasets incorporate more types of faults, but only contain up to one fault per scenario. On the other hand, the DP2 datasets contain up to three faults per scenarios, with each fault injected independently of any others.

The DXC11 datasets were generated six months after the DXC10 datasets. During this six month aging of the ADAPT EPS, its nominal behavior changed slightly (see Section 7.4), presenting a real-world challenge for a diagnostic system like ProDiagnose.

Simulations using ADAPT DP2 and DP1 scenarios were conducted using the DXC10 framework, which provides the sensor and command data to ProDiagnose, and records the output from ProDiagnose, including all metrics calculations [46]. All experiments involving the DXC10 framework with ProDiagnose were performed on a computer running an Intel Core2 Duo T6400 @ 2 GHz (using a single core) with 4GB RAM. The JVM used was version 1.6.0_22 (OpenJDK 1.10.8 64-bit).

In the following sections, we present experimental results using the following metrics [40,41,46]:

- **Detection Accuracy**: Percentage of scenarios that ProDiagnose correctly diagnosed. This includes diagnosing the correct faults for scenarios in which faults were injected (faulty), and not diagnosing any faults for scenarios in which no fault was injected (nominal).
- **False Positives Rate**: Percentage of scenarios in which ProDiagnose incorrectly diagnosed faults that were in fact not present.
- **False Negatives Rate**: Percentage of scenarios in which ProDiagnose incorrectly missed faults that were actually present.

7. Experiments

In this section we report on experiments with the ADAPT EPS (see Section 3). Expanding upon previous work [3,5,14,15], we include results on inference times and methods, and aging of the ADAPT electrical power system. We also expanded the number of datasets used for experimentation; we have no previous results for the DXC11 datasets.

7.1. Data and methods

All experimentation was performed using scenario data obtained from the ADAPT EPS (see Fig. 1, Section 3), more specifically the DXC10 and DXC11 datasets. The BN model of the ADAPT EPS was introduced in Section 5.3 (see Fig. 11).

Table 7 gives an overview of the datasets used in the DX competitions (DXC10 and DXC11). The DP1 datasets incorporate more types of faults, but only contain up to one fault per scenario. On the other hand, the DP2 datasets contain up to three faults per scenarios, with each fault injected independently of any others.

The DXC11 datasets were generated six months after the DXC10 datasets. During this six month aging of the ADAPT EPS, its nominal behavior changed slightly (see Section 7.4), presenting a real-world challenge for a diagnostic system like ProDiagnose.

Simulations using ADAPT DP2 and DP1 scenarios were conducted using the DXC10 framework, which provides the sensor and command data to ProDiagnose, and records the output from ProDiagnose, including all metrics calculations [46]. All experiments involving the DXC10 framework with ProDiagnose were performed on a computer running an Intel Core2 Duo T6400 @ 2 GHz (using a single core) with 4GB RAM. The JVM used was version 1.6.0_22 (OpenJDK 1.10.8 64-bit).

In the following sections, we present experimental results using the following metrics [40,41,46]:

- **Detection Accuracy**: Percentage of scenarios that ProDiagnose correctly diagnosed. This includes diagnosing the correct faults for scenarios in which faults were injected (faulty), and not diagnosing any faults for scenarios in which no fault was injected (nominal).
- **False Positives Rate**: Percentage of scenarios in which ProDiagnose incorrectly diagnosed faults that were in fact not present.
- **False Negatives Rate**: Percentage of scenarios in which ProDiagnose incorrectly missed faults that were actually present.
Table 8
Results from the 2010 DX competition (DXC10). ProDiagnose competed in both DP1 and DP2.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Training</th>
<th>Competition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP1</td>
<td>DP2</td>
<td>DP1</td>
</tr>
<tr>
<td>Detection Accuracy</td>
<td>92.31%</td>
<td>87.88%</td>
</tr>
<tr>
<td>False Positives Rate</td>
<td>0%</td>
<td>3.03%</td>
</tr>
<tr>
<td>False Negatives Rate</td>
<td>8.82%</td>
<td>11.54%</td>
</tr>
<tr>
<td>Mean Time To Detect</td>
<td>17.97 s</td>
<td>5.68 s</td>
</tr>
<tr>
<td>Mean Time To Isolate</td>
<td>72.27 s</td>
<td>37.84 s</td>
</tr>
</tbody>
</table>

- Mean Time To Detect: The time from fault injection ($t^*$) until any fault is detected. In the case of multiple fault injections, this time is in reference to the first fault injected. The detected fault may or may not be the correct fault.
- Mean Time To Isolate: The time from $t^*$ until the correct fault is diagnosed. In the case of multiple fault injections, each fault will have its own isolation time. When a fault is first detected, the actual fault type and part that is faulty may not be obvious. As the fault progresses, its behavior tends to stabilize, and the fault signature tends to become more apparent. See Section 4.2.2 for examples.

7.2. DX competition experiments and results

Here we present the results of ProDiagnose from the DXC10 competition, and compare them to the results using the DXC10 training dataset with which we configured ProDiagnose. The competition results were performed by the DXC10 committee at NASA Ames Research Center [40, 43].

For both the training dataset and competition dataset, ProDiagnose used the same configuration. This configuration used the following parameters for DP1: $c_{DD} = 300$, $c_{CO} = 80$ and $c_{FD} = 6$ (see Section 6). The same parameters were used for DP2, except that $c_{FD} = 5$.

Table 8 shows a comparison of the experimental results for the training and competition datasets for ProDiagnose. ProDiagnose won the DP2 tier in DXC10 [46], and as can be seen in Table 8, its accuracy was actually slightly higher for competition than for training. The higher accuracy was due to the much lower false negatives rate. ProDiagnose suffered a decrease in accuracy for DP1 by about 10%. While the false positives rate only increased by 1%, the false negatives rate jumped from 8.82% to 17.91%. This can be partly attributed to the competition datasets being recorded from the ADAPT EPS about 6 months after the training datasets. ProDiagnose is configured to suppress possible faults until it is quite certain that a fault is actually present, thus producing more false negatives when the nominal behavior of the physical EPS changes slightly. Note that this did not affect DP2, only DP1. This is because DP1 incorporates more fault types, notably subtle drift and intermittent faults, which require sensitive supplementary processing within ProDiagnose, and these supplemental algorithms are configured to reduce false positives (see Sections 6.5.2 and 6.5.3).

7.3. Inference method experiments

This experiment aimed to investigate the differences between the two ACE inference methods—MOP and SOP—discussed in Section 6. More specifically, since ProDiagnose is configured with MOP in mind, would changing the inference method to SOP and rerunning a set of scenarios affect any of the performance metrics? Using the DXC10 training dataset, we switched ProDiagnose to use the SOP method. All other configuration remained the same.

Looking at Table 9, notice that the accuracy for DP2 remained unchanged. The only DP2 metric that had a considerable change was the mean time to isolate. The reason for this difference in isolation time lies in the inference method, and we now provide some insight. When an answer to a query is of high certainty, the MOP (computed using MPE($e$)) and the SOP (computed using BEL($X,e$)) tend to be the same. However, when the situation is highly ambiguous and the ACE answer is highly uncertain, the MOP, when looking at the MPE($e$) for a single part, may not match the most likely state from BEL($X,e$), computed using SOP. This can result in isolation times fluctuating, as is the case with our DP2 results shown in Table 9.

---

21 For more information on the DXC10 competition, visit the following URL: http://www.phmsociety.org/competition/dxc/10.
22 Previous works only included training or competition datasets. Specifically, [3, 14] included experimental results from competition datasets, while [5, 15] included experimental results from training datasets.
23 The DXC11 training datasets were also recorded around the same time as the DXC10 competition dataset.
24 The initial parameters for both the intermittent and drift trackers are set per component or sensor, but for the intermittent tracker we chose unified parameters for all sensors and components. More specifically, the intermittent tracker was configured as follows: $t_{in} = [10, 350]$, $t_{IF} = [15, 120]$ and $c_{IF} = 2$. For more aggressive settings, $t_{in}$ and $t_{IF}$ could be set with lower min values.
25 This is not a reflection of inference times, which we discuss in Section 7.5, but rather an indication of the number of cycles needed to isolate a fault on average decreased (see Section 6).
Table 9
Experimental results using two methods of inference incorporated into the ACE inference engine. The MOP method takes the most probable explanation (MPE) whereas the SOP method uses marginal computations.

<table>
<thead>
<tr>
<th>Metric</th>
<th>MPE (MOP)</th>
<th>Marginals (SOP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP1</td>
<td>DP2</td>
<td>DP1</td>
</tr>
<tr>
<td>Detection Accuracy</td>
<td>92.31%</td>
<td>87.88%</td>
</tr>
<tr>
<td>False Positives Rate</td>
<td>0%</td>
<td>3.03%</td>
</tr>
<tr>
<td>False Negatives Rate</td>
<td>8.82%</td>
<td>11.54%</td>
</tr>
<tr>
<td>Mean Time To Detect</td>
<td>17.97 s</td>
<td>5.68 s</td>
</tr>
<tr>
<td>Mean Time To Isolate</td>
<td>72.27 s</td>
<td>37.84 s</td>
</tr>
</tbody>
</table>

Table 10
Experimental results from EPS aging. The DXC11 training datasets were recorded from the ADAPT EPS about 6 months after the DXC10 datasets. The same configuration of ProDiagnose was used for both the DXC10 and DXC11 datasets.

<table>
<thead>
<tr>
<th>Metric</th>
<th>DXC10</th>
<th>DXC11</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP1</td>
<td>DP2</td>
<td>DP1</td>
</tr>
<tr>
<td>Detection Accuracy</td>
<td>92.31%</td>
<td>87.88%</td>
</tr>
<tr>
<td>False Positives Rate</td>
<td>0%</td>
<td>3.03%</td>
</tr>
<tr>
<td>False Negatives Rate</td>
<td>8.82%</td>
<td>11.54%</td>
</tr>
<tr>
<td>Mean Time To Detect</td>
<td>17.97 s</td>
<td>5.68 s</td>
</tr>
<tr>
<td>Mean Time To Isolate</td>
<td>72.27 s</td>
<td>37.84 s</td>
</tr>
</tbody>
</table>

With the above reasoning in mind, we now consider the DP1 experimental results. While the false positives rate stayed the same, the false negatives rate increased to 14.71% for SOP from 8.82% for MOP. This higher false negatives rate for SOP compared to MOP means the following. Using SOP resulted in the health state for some faulty parts being incorrectly diagnosed as healthy while the MPE (in MOP) resulted in a correct diagnosis of not healthy. This follows from the higher uncertainty of faulty diagnoses when using SOP.

Both detection and isolation times for DP1 also increased for SOP, and the reasoning behind this correlates to the higher false negatives rate observed. In the face of higher uncertainty observed for SOP, a faulty state will at times take longer to properly diagnose, and this is captured in terms of the detection and isolation times from Table 9.

Overall, while the DP2 results were similar for both MOP and SOP, the DP1 results showed an advantage with MOP, especially in terms of the false negatives rate.

7.4. ADAPT EPS aging experiments

The goal of the aging experiment was to investigate how robust ProDiagnose is with respect to changes in the nominal behavior of an electrical power system as it ages naturally. We are not aware of similar robustness experiments in past research.

The DXC11 training datasets were recorded about 6 months after the DXC10 datasets. This means that we were not simply simulating a natural aging of ADAPT, but rather used datasets that physically captured this natural drift. For this experiment, we ran the training datasets from the DXC11 competition, and compared the results with that from the DXC10 training datasets. ProDiagnose used the same configuration for both datasets.

Looking at Table 10, several interesting trends emerge. While DP1’s detection accuracy drops about 10%, DP2’s accuracy actually increases about 1%. The false positives rates for both DP1 and DP2 increased, which is to be expected given a natural drift in ADAPT’s nominal behavior over 6 months. For DP2, on the other hand, the false negatives rate dropped from 11.54% to 4.08%. One interesting consequence of natural drift in sensor readings due to aging is that it helps to bring out underlying behavior present in certain faults that may otherwise go undiagnosed, such as increasing the offset magnitude, \(\Delta p\) (see Section 4.2.2), for offset faults of small magnitude. These types of faults are common for DP2, and ProDiagnose sometimes struggled with them in DXC10.26

Overall, while the false positives rate increased for both DP1 and DP2, the aging of ADAPT helped to bring out underlying faulty behavior which lowered the false negatives rate in DP2 substantially.

---

26 CH nodes can become quite expensive as their quantity increases, and thus were not attached to every sensor in the DP2 Bayesian network. As a consequence, many small-magnitude offset faults went undiagnosed in DXC10 DP2 scenarios.
Table 11
Benchmarking results of the ACE inference engine. A Java Benchmarking API is integrated into ProDiagnose which records the inference times of ACE during a designated point of time in the simulation.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Mean (μs)</th>
<th>Standard deviation (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First cycle</td>
<td>1200th cycle</td>
</tr>
<tr>
<td>(DP2) Exp_1201</td>
<td>328.122</td>
<td>349.399</td>
</tr>
<tr>
<td>(DP2) Exp_1202</td>
<td>338.366</td>
<td>333.060</td>
</tr>
<tr>
<td>(DP2) Exp_1158f</td>
<td>341.406</td>
<td>334.941</td>
</tr>
<tr>
<td>(DP1) Exp_1127_008f</td>
<td>199.267</td>
<td>189.589</td>
</tr>
<tr>
<td>(DP1) Exp_1175</td>
<td>199.356</td>
<td>189.787</td>
</tr>
</tbody>
</table>

7.5. ACE inference benchmarks

The goal of the experiments in this section was to benchmark the computational speed of the ACE inference engine, as used in ProDiagnose (see Section 6).

The benchmarking was performed using Brent Boyer’s Java benchmarking API, on a computer running an Intel Core2 Duo T6400 @ 2 GHz (using a single core) with 4GB RAM. The JVM used was version 1.6.0_22 (OpenJDK 1.10.8 64-bit). This API was picked due to the accuracy of the results it gives. We integrated this API directly into ProDiagnose. When called, the Benchmarking API will call ACE repeatedly over the span of approximately 10 seconds. During each call, ACE will use the same evidence provided from ProDiagnose. Thus, the ADAPT simulation essentially halts while the benchmarking API is running. As we will see by the results, ACE is called 10,000’s of times during this 10 second interval.

During each call to ACE, the total time taken for inference was recorded. After the Benchmarking API is finished, the mean, denoted as $\bar{\tau}$, and standard deviation are retained, and these are presented here. Each simulation is executed three times per scenario: once at the beginning of the scenario (first call to ACE), once in the middle of the scenario (1200th call to ACE), and once within the last minute of the scenario (2000th call to ACE).

Simulations using scenarios from both the DXC10 DP1 and DP2 datasets are represented in Table 11. For a given configuration (DP1 or DP2), $\bar{\tau}$ is very consistent across each scenario and across all scenarios. Specifically, $\bar{\tau}$ ranges from 328 μs to 349 μs for DP2, and from 188 μs to 199 μs for DP1. Standard deviation is not as consistent, due to extreme outliers in almost all the simulation runs. We speculate that this is due, at least partly, to the use of Java, and the overhead the JVM garbage collector can impose on a particular simulation run. However, thanks to the sheer number of samples generated in the benchmarks, these outliers did not substantially affect $\bar{\tau}$. Notice that $\bar{\tau}$ values for DP1 are much lower than for DP2. This is due to the much larger arithmetic circuit needed to evaluate DP2, which represents the full ADAPT system. DP1’s AC, on the other hand, represents a relatively small subset of the ADAPT EPS. The decrease in $\bar{\tau}$ magnitude is roughly linear in the decrease in AC size.

Generally, the computational cost is very low, and is proportional to the size of the AC model.

8. Conclusion

Bayesian networks have been successfully applied in a broad range of applications, and are supported by numerous exact and inexact inference algorithms that compute marginals and most probable explanations (MPEs) [6,10,12,21,22]. To our knowledge, this is among the first efforts that develops, using static Bayesian networks compiled to arithmetic circuits, a comprehensive approach to fault diagnosis in dynamic and hybrid domains. The approach, which we call ProDiagnose, performs fault diagnosis under varying fault persistency (both persistent and intermittent faults), fault progression (both abrupt and drift faults), and fault cardinality (both discrete and continuous faults).

Using a large number of scenarios from two configurations (DP1 and DP2) of a NASA electrical power system, we have experimentally validated the approach, which includes training and competition results from the DXC10 competition, experiments using different inference methods for ProDiagnose, aging experiments, and inference benchmarking. More specifically, we showed that ProDiagnose kept a high level of accuracy in the DXC10 competition compared to the DXC10 training sets, even showing a higher level of accuracy in the competition for DP2. Using both MOP and SOP inference methods for ProDiagnose resulted in MOP showing a higher accuracy and lower fault isolation times for DP1, with DP2 showing no difference in accuracy, but lower isolation times when using SOP. The aging experiments using the DXC10 and DXC11 datasets showed that ProDiagnose could remain accurate as a system ages over time, with DP2 even showing a slight increase in accuracy from DXC10 to DXC11. Benchmarking ProDiagnose’s inference engine yields both fast and consistent mean times, with DP1 in the range of 188–199 μs, and DP2 in the range of 328–349 μs across scenarios. Finally, ProDiagnose has also been the top performer in three of the four international diagnostics competitions where it recently participated, arranged at the DX09 and DX10 conferences.

28 More information on these international diagnostics competitions can be found here: http://www.dx-competition.org/.
