Anticipatory Troubleshooting

Netanel Hasidi and **Roni Stern** and **Meir Kalech**
Ben Gurion University of the Negev
Be'er Sheva, Israel
{hasidi,sternron,kalech}@bgu.ac.il

Shulamit Reches
Jerusalem College of Technology
Jerusalem, Israel
shulamit.reches@gmail.com

Abstract

Troubleshooting is the process of diagnosing and repairing a system that is behaving abnormally. Diagnostic and repair actions may incur costs, and traditional troubleshooting algorithms aim to minimize the costs incurred until the system is fixed. We propose an anticipatory troubleshooting algorithm, which is able to reason about both current and future failures. To reason about failures over time, we incorporate statistical tools from survival analysis that enable predicting when a failure is likely to occur. Incorporating this prognostic information in a troubleshooting algorithm enables (1) better fault isolation and (2) more intelligent decision making in which repair actions to employ to minimize troubleshooting costs over time. This paper was accepted to the International Joint Conference on Artificial Intelligence (IJCAI) 2016.

1 Introduction

System failures are prevalent in practically all the engineering fields, including automobiles, robots, information systems, and computer hardware. As systems become more complex, failures often become more common and maintenance costs tend to increase. Thus, automated diagnosis has been studied in the artificial intelligence field for several decades, with substantial progress and successful applications in spacecrafts [Williams and Nayak, 1996], satellite decision support systems [Feldman *et al.*, 2013], automotive industry [Struss and Price, 2003] and spreadsheets [Jannach and Schmitz, 2014]. The output of diagnosis algorithms is a set of possible diagnoses, where each possible diagnosis is an explanation of the observed system failure.

While **diagnosis**, and in particular root-cause analysis, is the task of understanding what has happened in the past that has caused an observed failure, **prognosis** is the task of predicting what will happen in the future, and in our context when will future failures occur. In parallel to the vast literature on automated diagnosis, there has been much work on developing prognosis techniques for estimating the remaining useful life of components in a system. In particular, survival analysis is a sub-field of statistics in which various methods have been developed to generate **survival curves** of components, which are curves that plot the likelihood of a component to survive (not fail) as a function of the components usage or age [Miller Jr, 2011]. Sophisticated Machine Learning techniques have been shown to be able to learn such survival curves from data [Eyal *et al.*, 2014].

Consider the following illustrative example, in which a car does not start. The mechanic inspecting the car observes that the water level in the radiator is low. This suggests that the radiator is not functioning well, which is a possible explanation – a diagnosis – for why the car does not start. However, alternative diagnoses are that the ignition system is faulty or the battery is empty. Clearly, considering the age of the battery and the survival curve of batteries of the same type can provide valuable input to the mechanic in deciding the most likely diagnosis and consequent next troubleshooting action.

The **first main contribution** of this work is in showing how prognosis and diagnosis, and in particular survival curves and automated diagnosis algorithms, can be integrated effectively. Practically, we propose an improved diagnosis algorithm that considers both diagnostic information about the relation between sensor data and faults, as well as the likelihood of each component to fail given its age, obtained from the corresponding survival curves.

Beyond diagnosis, we show that the integration of survival curves into the model assists also to the troubleshooting process. **Troubleshooting** is the process of diagnosing and repairing an observed failure. Diagnostic and repair actions may incur costs, such as the time spent in observing internal components and the monetary cost of purchasing a new component to replace a faulty one. Troubleshooting algorithms aim to minimize the costs incurred until the system is fixed.

The **second main contribution** is to use prognosis tools, and in particular, survival curves, to develop a troubleshooting algorithm that aims to minimize current troubleshooting costs and future maintenance costs. These maintenance costs includes costs due to future failures, which would require additional troubleshootings and perhaps system downtime. We refer to this type of troubleshooting, where future costs are also considered, as **anticipatory troubleshooting**, and propose an effective anticipatory troubleshooting algorithm. In particular, the specific dilemma our troubleshooting algorithm addresses is how to choose the most appropriate repair action, given a component that is identified as faulty. For ex-

ample, fixing a faulty component may be cheaper than replacing it with a new one. On the other hand, a new component is less likely to fail in the near future. Our anticipatory troubleshooting algorithm leverages available survival curves to choose the appropriate repair action intelligently. We demonstrated experimentally the benefit of our approach on benchmark systems modeled by a Bayesian network, showing significant gains.

2 Related Work

Many works on automated troubleshooting are based on the seminal work of Heckerman et al. [1995] on Decision Theoretic Troubleshooting (DTT). Our paper can be viewed as a generalization of DTT by adding prognosis consideration. A decision theoretic approach integrating planning and diagnosis was applied to a real-world troubleshooting application [Warnquist *et al.*, 2009; Pernestål *et al.*, 2012]. In their setting a sequence of actions may be needed to perform repairs. For example, a vehicle may need to be disassembled to gain access to its internal parts. To address this problem, they used a Bayesian network for diagnosis and the AO* algorithm [Nilsson, 1982] as the planner. Torta et al. [2014] proposed the use of abstractions to improve the efficiency of troubleshooting. Friedrich and Nedjl [1992] proposed a troubleshooting algorithm aimed at minimizing the breakdown costs, a concept that corresponds roughly to a penalty incurred for every faulty output in the system and for every time step until the system is fixed. The novelty of our work is two-fold. First, none of these previous works incorporated prognosis estimates into the troubleshooting algorithm (as we do in Section 4.1). Second, our anticipatory troubleshooting model and algorithm directly attempt to minimize costs incurred due to current and future failures.

A related line of work is in prognostics, where diagnostic information is used to provide more accurate remaining useful life estimations [Ferreiro *et al.*, 2012; Tobon-Mejia *et al.*, 2012]. There is also work in prognostic on Prognostic Decision Making, where scheduling maintenance activities is done intelligently by using prognostic information [Balaban and Alonso, 2012]. By contrast, we aim to improve decision making for fixing a current fault, but consider also future faults. Somewhat similar is the work of McNaught and Zagorecki (2009), who considered several possible repair actions and their impact on future faults by unfolding a dynamic Bayesian network. However, they did not propose an actual method to choose which action to perform, as we do.

3 Troubleshooting

A system is composed of a set of components, denoted *COMPS*. A component $C \in$ *COMPS* is either healthy or faulty, denoted by the health predicate $h(C)$ or $\neg h(C)$, respectively. The **state of a system**, denoted ξ, is a conjunction of health literals defining for every component whether it is healthy or not. A troubleshooting agent is an agent capable of performing sensing and repair actions. The **agents belief about the state of the system**, denoted B, is a conjunction of health literals. We assume that the agents knowledge is correct, i.e., if $h(C) \in B \rightarrow h(C) \in \xi$. The agents belief, however, may be incomplete, i.e., there may exists a $C \in$ *COMPS* such that neither $h(C)$ nor $\neg h(C)$ is in B. A troubleshooting problem arises if the system is identified as faulty, e.g., by some fault detection mechanism. We assume that such a mechanism exists, revealing to the agent whether the system is faulty or not. In this work we focus on the **single-fault** case, i.e., where a single component is faulty.

Action	System state (ξ)	Agent's belief B
Start	$\{\neg h(C_1), h(C_2), h(C_3)\}$	$\{h(C_3)\}$
$Sense_{C_2}$	$\{\neg h(C_1), h(C_2), h(C_3)\}$	$\{h(C_3), h(C_2)\}$
$Sense_{C_1}$	$\{\neg h(C_1), h(C_2), h(C_3)\}$	$\{h(C_3), h(C_2), \neg h(C_1)\}$
$Repair_{C_1}$	$\{h(C_1), h(C_2), h(C_3)\}$	$\{h(C_3), h(C_2), h(C_1)\}$

Table 1: An example of a troubleshooting process

An action of the troubleshooting agent is a transition function, accepting and potentially modifying both system state ξ and agent's belief B. We consider two types of actions: **sense** and **repair**. Each action is parametrized by a single component, where $Sense_C$ checks if C is healthy or not, and $Repair_C$ results in C being healthy.[1] Formally, applying $Sense_C$ does not modify ξ, and updates B by adding $h(C)$ if $h(C) \in \xi$ or adding $\neg h(C)$ otherwise. Similarly, applying $Repair_C$ adds $h(C)$ to both B and ξ, and removes $\neg h(C)$ from B and ξ if it was there.

Definition 1 (Troubleshooting Problem (TP)) *A TP is defined by the tuple P=$\langle$COMPS$, \xi, B, A\rangle$ where (1) COMPS is the set of components in the system, (2) ξ is the state of the system, (3) $B \subseteq \xi$ is the agent's belief about the system state, and (4) A is the set of actions the troubleshooting agent is able to perform. A TP arises if $\exists C\ \neg h(C) \in \xi$. A solution to a TP is a sequence of actions that results in a system state in which all components are healthy.*

A troubleshooting algorithm (TA) is an algorithm for guiding a troubleshooting agent faced with a TP. TAs are iterative: in every iteration the TA accepts the agent's current belief B as input and outputs a sense or repair action for the troubleshooting agent performs. A TA halts when the sequence of actions it outputted forms a solution to the TP, i.e., when the system is fixed. The solution outputted by a TA π to a TP P is denoted by $\pi(P)$. Both sense and repair actions incur a cost. The cost of an action a is denoted by $cost(a)$. The cost of solving P using π, denoted by $cost(\pi, P)$, is the sum of the costs of all actions in $\pi(P)$: $cost(\pi, P) = \sum_{a \in \pi(P)} cost(a)$. TAs aim to minimize this cost.

Consider the mentioned earlier car diagnosis example, in which there are three relevant components that may be faulty: the radiator (C_1), the ignition system (C_2), and the battery (C_3). Assume that the radiator is the correct diagnosis, meaning the radiator is really faulty, and the agent knows that the battery is not faulty. The corresponding system state ξ and agent's belief B are represented by: $\xi = \{\neg h(C_1), h(C_2), h(C_3)\}$ and $B = \{h(C_3)\}$. Table 1 lists a solution to this TP, in which the agent first senses the ignition system, then the radiator, and finally repairs the radiator. Formally, $\pi(P) = \{Sense_{C_2}, Sense_{C_1}, Repair_{C_1}\}$. If the cost

[1] See Stern et al. [2016] for a discussion on actions that apply to a batch of components.

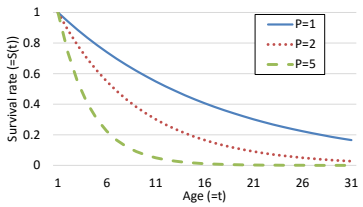


Figure 1: An example of exponential survival curves.

of sense is one and the cost of repair is five, then the troubleshooting costs of this solution is 1+1+5=7.

4 Troubleshooting with Survival Functions

If the cost of sense actions is much smaller than the cost of repair actions, then an intelligent troubleshooting algorithm would only repair components that were first identified as faulty as a result of a sense action. This simplifies the troubleshooting process: perform sense actions on components until a faulty component is found, and then repair it. The challenge is which component to sense first.

To address this challenge, efficient troubleshooting algorithms use a diagnosis algorithm (DA). A DA outputs one or more diagnoses, where a diagnosis is a hypothesis about which components are faulty. Moreover, many DAs output for each diagnosis ω the likelihood that it is correct, denoted $p(\omega)$. These diagnoses likelihoods can be aggregated to provide an estimate of the likelihood that each component is faulty, denoted $p(C)$ [Stern *et al.*, 2015]. A reasonable troubleshooter can then choose to sense first the component most likely to be faulty.

Many DAs have been proposed in the literature. Most effective DAs use some prior knowledge about the diagnosed system to provide accurate diagnoses. Model-based diagnosis (MBD) is a classical approach to diagnosis in which an existing model of the system together with observations of the system behavior, are used to infer diagnoses. Some MBD algorithms assume a system model that represents the system behavior using propositional logic and use logical reasoning to infer diagnoses that are consistent with system model and observations [Feldman *et al.*, 2010; Williams and Ragno, 2007; De Kleer and Williams, 1987]. In general, most MBD algorithms implicitly assume that the system model represents the relation between the system inputs (including sensors) and outputs, and the components behavior. In this work we use a DA based on a Bayesian network (BN) that represents the probabilistic dependency between observations and the system health state. Next, we show how techniques from survival analysis allow augmenting such models with information about the age of each component and its implication on the likelihood of components to be faulty.

4.1 Integrating Survival Analysis into a DA

Every component C is associated with an age Age_C. Let T_C be a random variable representing the age in which C will fail. A **survival function** for C, denoted $S_C(t)$, is the probability that C will survive until the age t, meaning it will not fail before age t. Formally: $S_C(t) = Pr(T_C \geq t)$. Survival functions can be obtained by analysis of the physics of the corresponding system or learned from past data [Eyal *et al.*, 2014]. Figure 1 illustrates three possible survival curves generated by an exponential decay function $e^{-\lambda \cdot t}$, where λ is a parameter and t is the age (the x-axis). The y-axis represents the probability that a component will survive (= not fail) t time units (e.g., months). The three curves plotted in Figure 1 correspond to three values of the λ parameter.

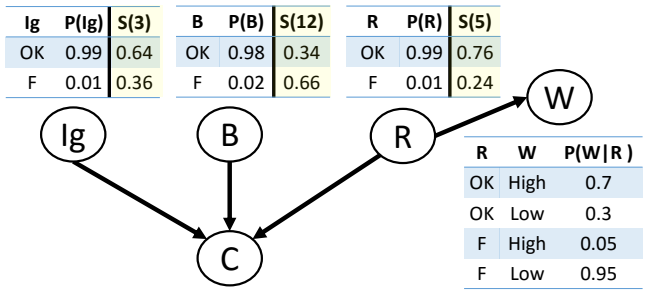
The challenge is how to compute the probability that a component C caused a system failure given its age and survival function. In most systems, faulty components *fail intermittently*, i.e., a component may be faulty but act normally. Thus, the faulty component that caused the system to fail may have been faulty even before time t. Therefore, we estimated the probability of a component C of age Age_C to have caused the system failure by the probability that it has failed anytime before the current time. This probability is directly given by $1 - S_C(Age_C)$, denoted by $F_C(Age_C)$.

Thus, for a given component C we have two estimates for the likelihood that it is correct: one from the MBD algorithm ($p(C)$) and one from its survival curve ($F_C(Age_C)$). The MBD algorithm's estimate is derived from the currently observed system behavior or knowledge about the system's structure. The survival curve estimate is derived from knowledge about how such components tend to fail over time. We propose to combine these estimates to provide a more accurate and more informed diagnostic report.

A simplistic approach to combine these fault likelihood estimates is by using some weighted linear combination, such that the weights are positive and sum up to one. However, we argue that these estimates are fundamentally different: $F_C(Age_C)$ is an estimate given **a-priori** to the actual fault, while $p(C)$ is computed by the MBD algorithm for the specific fault at hand, taking into consideration the **currently observed** system behavior. Indeed, MBD algorithms often require information about the prior probability of each component to be faulty when computing their likelihood estimates [De Kleer and Williams, 1987; González-Sanchez *et al.*, 2011; Zamir *et al.*, 2014; Mengshoel *et al.*, 2010]. However, these priors are often set to be uniform, although it has been shown that setting such priors more intelligently can significantly improve diagnostic accuracy [Elmishali *et al.*, 2016]. Therefore, we propose to use the fault likelihood estimation given by the survival curves as priors within the likelihood estimation computation done by the MBD algorithm.

Specifically, we experimented with an MBD that computes diagnoses by applying inference on a BN [Mengshoel *et al.*, 2010]. The BN contains both health variables and other variables such as sensor readings. The values of the observable variables are set, and then the marginal of each health variable is computed by applying an inference algorithm on the BN. The Bayesian reasoning done by the inference algorithm requires a prior probability. We propose to use $S_C(Age_C)$ as this prior probability (and normalize the fault probability over the remaining probability sum). Other ways to integrate survival curves in an MBD are also possible, and the key contribution is that doing so is beneficial.

Consider our running example of a car that does not start. Figure 2 depicts a possible BN representing this example. Nodes Ig, B, and R correspond to the health variables for



Ig	P(Ig)	S(3)
OK	0.99	0.64
F	0.01	0.36

B	P(B)	S(12)
OK	0.98	0.34
F	0.02	0.66

R	P(R)	S(5)
OK	0.99	0.76
F	0.01	0.24

R	W	P(W|R)
OK	High	0.7
OK	Low	0.3
F	High	0.05
F	Low	0.95

Figure 2: A simple BN for our car diagnosis running example.

the ignition, battery, and radiator, respectively. Now W correspond to the water level variable, and C correspond to the observation that the car not starting. The Conditional Probability Tables (CPTs) for all nodes except C are also displayed in Figure 2. The value of C deterministically depends on Ig, B, and R: only if all components are healthy can the car start. Modeling such dependency (a logical OR) in a BN is trivial. Note that we disallow multiple faults (these are mapped to a "N/A" value of C). Now, assume that the car does not start (C=False) and the water level is low (W=Low). Given this evidence we apply Bayesian reasoning to get the likelihood of each component to be faulty. In our case, the likelihood of Ig, B, or R to be faulty is 0.16, 0.33, and 0.52, respectively. Thus, a troubleshooter would sense first R.

However, assume that the ages of the ignition (Ig), battery (B), and radiator (R) are 3, 12, and 5, respectively, and that they all follow an exponential survival curve $e^{-0.09 \cdot t}$. Thus, according to the components age and survival curves the probability of Ig, B, and R to be faulty are 0.24, 0.66, and 0.36, respectively. Setting these probabilities instead of the original health nodes' priors is shown in Figure 2 in the $S(X)$ columns in the CPTs. Setting these priors dramatically affects the result of the Bayesian reasoning, where now the probability of Ig, B, and R to be faulty is 0.16, 0.56, and 0.28, respectively. As a result, a troubleshooter that is aware of both BN and survival curves would choose to sense the battery (rather than the radiator).

5 Anticipatory Troubleshooting

The main benefit of using survival functions in the context of troubleshooting is in the ability to reason about future failures, with the goal of **minimizing troubleshooting costs over time**. Formally, let $[0, T_{limit}]$ be the time range in which we aim to minimize troubleshooting costs. During this time range, components in the system may fail. When the system fails, a troubleshooting process is initiated, performing sense and repair actions until the system is fixed. The target function we wish to minimize is the sum of costs incurred due to actions performed by the troubleshooting agent in the time range $[0, T_{limit}]$. [2] We refer to this sum of troubleshooting costs as the **long-term troubleshooting cost**, and call a troubleshooting algorithm that aims to minimize this cost an **anticipatory troubleshooting algorithm**.

[2]Note that we exclude from this process the task of fault detection, i.e., we assume that the troubleshooter is given a system that is not working properly and needs to be repaired.

When there is only a single sense action and a single repair action, there is no difference between an anticipatory troubleshooting and a troubleshooting algorithm only aiming to minimize the current troubleshooting costs. The difference between traditional troubleshooting and anticipatory troubleshooting is meaningful when there are multiple repair actions. In other words, after the troubleshooting algorithm identified which component is faulty, the troubleshooter needs to decide which repair action to use to repair it. Next, we describe such a setting, where there are two possible repair actions: **Fix** and **Replace**.

5.1 Fix vs. Replace

Applying a *Replace(C)* action means that the troubleshooting agent replaces C with a new one. Applying *Fix(C)* action means that the troubleshooting agent fixes C without replacing it. Both fix and replace are repair actions, in the sense that after performing them the component is healthy and the agent knows about it, i.e., replacing $\neg h(C)$ with $h(C)$ in both the system state and the agent's belief.

However, we expect Fix and Replace to differ in two realistic aspects. First, Fix is expected to be cheaper than Replace. Second, after replacing a component its ability to survive is expected to be significantly higher than that after it has been fixed, since the replaced component is new. To formalize this, let $S_C(t, Age_C)$ be the survival curve of C after it was fixed at age Age_C, i.e., the probability of C to survive t time units after it was fixed, given that it was fixed at age Age_C.

$$S_C(t, Age_C) = Pr(T_C \geq t + Age_C | C \text{ fixed at age } Age_C)$$

We call such a survival function an **after-fix survival** function. Formally, the expected differences relations between fix and replace are:

$$\forall C \in \textit{COMPS} : cost(\textit{Fix(C)}) < cost(\textit{Replace(C)}) \quad (1)$$

$$\forall t \in [0, T_{limit}] \; \forall C \in \textit{COMPS} : S_C(t, Age_C) < S_C(t) \quad (2)$$

Intuitively, fixing a faulty component is cheaper, but may result in future faults being more frequent. This embodies the main dilemma in anticipatory troubleshooting: weighing current troubleshooting costs (Fix is preferable) against potential future troubleshooting costs (Replace is preferable).

5.2 Choosing the Appropriate Repair Action

An exhaustive and almost optimal approach to choose which repair action to perform is to discretize the time range $[0, T_{limit}]$, model the problem as a Markov Decision Problem (MDP), and apply an off-the-shelf MDP solver, as follows.

Discretization. The time limit $[0, T_{limit}]$ is partitioned to a non-overlapping set of equal-sized time ranges $T = \{T_0, \ldots, T_n\}$. Each T_i is referred to as a time step, and let Δt be the size of each time step.

MDP modeling. An MDP is defined by a state space $\mathcal{S}$, a set of actions $\mathcal{A}$, a reward function $R(s, a)$, and a transition function $Tr(s, a, s')$. A state in our state space is defined by a tuple $s = \langle T_i, C, \textit{Curves}, \textit{Ages} \rangle$, representing a state in which component C was diagnosed as faulty at time step T_i, where *Curves* and *Ages* are vectors representing the survival curves and ages of all components in *COMPS*. C can be null, representing a state in which no component was faulty at time

T_i. Recall that we only consider single fault scenarios, i.e, at most one component is fault at every time step. States for time T_{n+1} are terminal states. The set of actions $\mathcal{A}$ consists of three actions: *Replace(C)*, *Fix(C)*, and *no-op*. *no-op* represents not doing any action. The reward function $R(s, a)$ is minus the cost of the executed action, where the *no-op* action costs zero. The state transition function is as follows. After any action, a state for time step T_i will transition to a state for T_{i+1}. The MDP transition function $Tr(s, a, s')$, which is a function that returns the probability of reaching state s' after performing action a at state s, is defined as follows. Let $s = \langle T_i, C, \mathit{Curves}, \mathit{Ages}\rangle$ and $s' = \langle T_j, C', \mathit{Curves}', \mathit{Ages}'\rangle$. The values of T_j, *Curves*$'$, and *Ages*$'$, are set deterministically by s and a: $T_j = T_{i+1}$, *Curves*$'$ is only updated after a *Fix(C)* action (replacing C's survival function with its after-fix curve), and *Ages*$'$ consists of all components being older by one time step, except for when C is replaced (in which case the age of C is set to zero). The uncertainty in state transition is which component, if any, will be faulty in the next time step. Let $S_{C'}$ and $Age_{C'}$ be the survival curve and age of C' according to *Curves*$'$,*Ages*$'$. The probability that C' will fail at a specific time range T_j given its survival curve is:

$$Pr(T_{C'} \in T_j) = S_{C'}(Age_{C'} - \Delta t) - S_{C'}(Age_{C'})$$

which is a standard computation in survival analysis: the probability of surviving before T_j (when the age of C' was $Age_{C'} - \Delta t$) minus the probability of surviving until T_j (when the age of C' is $Age_{C'}$).

Solving the MDP. The state space of this MDP is exponential in the number of time steps reasoned about (n). Textbook MDP algorithms, such as Value Iteration and Policy Iteration, and AO* [Nilsson, 1982], will not work. Sophisticated sampling-based approaches, such as real-time dynamic programming (RTDP) [Barto *et al.*, 1995] may be applicable.

In our experiments we implemented a simple decision rule that roughly corresponds to reasoning about a single level of this MDP state space. We call this decision rule *Decision Rule 1* (*DR1*), and explain it next. Let $C_{replace} = cost(\mathit{Replace(C)})$, $C_{fix} = cost(\mathit{Fix(C)})$, and T_{left} be the time left until T_{limit}. Following *DR1* is to replace a faulty component C iff the following inequality holds:

$$C_{replace} + (1 - S_C(T_{left})) \cdot C_{replace} \leq C_{fix} + (1 - S_C(T_{left}, Age_C)) \cdot C_{replace} \quad (3)$$

DR1 has the following property.

Proposition 1 *DR1 is optimal if the following holds: (1) a component will not fail more than twice in the time range* $[0, T_{limit}]$*, (2) a component can be fixed at most once, (3) a replaced component will not be fixed in the future, and (4) components fail independently.*

Proof of proposition 1 is omitted due to space constraints. *DR1* is not optimal when the assumptions in Proposition 1 do not hold, but it can still be effective, as shown below.

6 Experimental Results

To evaluate the proposed algorithms, we performed two sets of experiments: "one-shot" experiments, in which a single TP is solved (evaluates the method in Section 4.1), and "long-term" experiments, in which troubleshooting costs are accumulated (evaluates the method in Section 5.2).

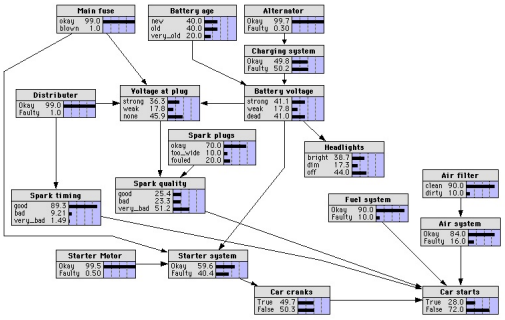


Figure 3: BN for system $S2$.

6.1 Benchmark Systems

The experiments were performed over two systems, modeled using a Bayesian network (BN) following the standard use of BN for diagnoses [Choi *et al.*, 2011]. The first system, denoted $S1$, represents a real world Electrical Power System. The BN was generated automatically from formal design [Mengshoel *et al.*, 2010] and is publicly available (`reasoning.cs.ucla.edu/samiam/tutorial/system.net`). It has 26 nodes, of which 6 are health nodes. The second system, denoted $S2$, is the "CAR_DIAGNOSIS_2" network from the library of benchmark BN made available by Norsys (`www.norsys.com/netlib/CarDiagnosis2.dnet`). The system represents a network for diagnosing a car that does not start, based on spark plugs, headlights, main fuse, etc. It contains 18 nodes. From these nodes, 7 nodes are health nodes. Figure 3 shows the graphical representation of $S2$.

6.2 Survival Curves and Component Ages

Unfortunately, we did not have real data for the above systems from which to generate survival curves. Therefore, we used a standard exponential curve (defined earlier in this paper and illustrated in Figure 1) with $\lambda = 0.09$. Exponential curves are fundamental parametric models used in the survival analysis literature [Ibrahim *et al.*, 2005].

We set the age of each component to be Age_{init} plus a random number between zero and Age_{diff}, where Age_{init} is a constant set arbitrarily to 0.3 and Age_{diff} is a parameter we varied in our experiments. The purpose of the Age_{diff} parameter is to control the possible impact of considering the components' survival functions: a small Age_{diff} results in all components having almost the same age, and thus the survival curves do not provide significant information to distinguish between which component is more likely to be faulty.

6.3 One-Shot Experiments

In this set of experiments we generated random TPs (details below) and compared the performance of four TAs: (1) **Random**, which chooses randomly which component to sense, (2) **BN-based**, which chooses to sense the component most likely to be faulty according to the BN, (3) **Survival-based**, which

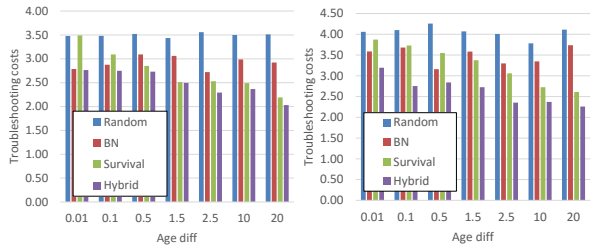


Figure 4: One-shot exp. results, for $S1$ (left) and $S2$ (right).

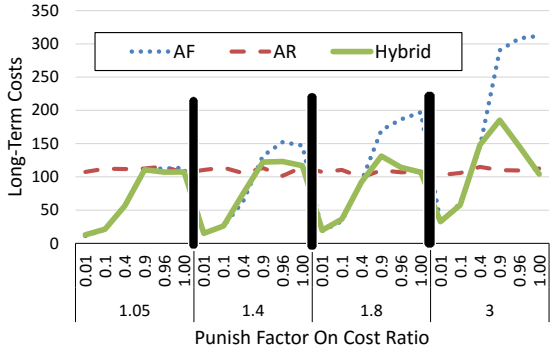


Figure 5: Long-term exp. results, for $S1$.

chooses to sense the component most likely to be faulty according to its survival curve and age, and (4) **Hybrid**, which chooses to sense the component most likely to be faulty taking into consideration both BN and survival curve, as proposed in Section 4.1. Performance of a TA was measured by the troubleshooting costs incurred until the system is fixed. Since we only considered single fault scenarios, we omitted the cost of the single repair action performed in each of these experiments, as all algorithms had spent this cost.

Each TP was generated with a single faulty health node as follows. The value of non-health nodes in the BN that do not depend on any other node were set randomly according to their priors. We refer to these nodes as control nodes. Then, the age of each component was set as mentioned above, i.e., by sampling uniformly in the range $[Age_{init}, Age_{init}+Age_{diff}]$. Then, the CPT of every health node was modified to take into account the survival curve as mentioned in Section 4.1, i.e., the prior of being healthy was set to $S_C(Age_C)$. Next, we computed the marginal probability of each component to be faulty in this modified BN, and chose a single component to be faulty according to these computed probabilities. Finally, we sampled from the BN values for all remaining nodes (nodes that are not control or health node), setting the values of the already set nodes. These nodes are called the sensor nodes, and a subset of them were revealed to the DA.

Figure 4 shows the troubleshooting cost for each of the algorithms, for different values of the Age_{diff} parameters, for $S1$ and $S2$. All results are averaged over 50 instances. Several trends can be observed. First, the proposed Hybrid TA outperforms all baseline TAs, thus demonstrating the importance of considering both survival curves and MBD. Second, as Age_{diff} grows, the performance of Survival improves, since the components' age differ more, and thus considering it is more valuable. When Age_{diff} is minimal, the performance of Survival is similar to Random and worse than BN. BN is better, since we provide it with evidence – the values of some sensor nodes (in the case of $S1$ we revealed 9 sensor nodes and for $S2$ we revealed 2 sensor nodes). We also experimented with different numbers of revealed nodes. As expected, revealing more nodes improves the performance of both BN and Hybrid. In conclusion, these results demonstrate that Hybrid is more robust than both Survival and BN, and is either equal or outperforms them across all varied parameter.

6.4 Long-Term Experiments

In this set of experiments we generated random TPs over a period of 28 months (i.e., T_{limit}=28), choosing when each component fails according to its survival function (details below). In each experiment we used one of the following TAs to solve the TPs that arise: (1) **Always Fix (AF)**, in which faulty components are repaired using the Fix action, (2) **Always Replace (AR)**, in which faulty components are repaired using the Replace action, and (3) **Hybrid**, in which *DR1* (Section 5.2) was used to choose the appropriate repair action. The performance of each algorithm is measured by the sum of troubleshooting costs incurred when solving all the TPs that arose. Since the focus of these experiments is to study the Fix vs. Replace dilemma, we omitted the costs incurred due to Sense action, and only measured the cost the repair action used in every troubleshooting session, i.e., $C_{replace}$ or C_{fix}.

To sample when a component will fail after it was fixed, and to compute the Hybrid TA, we require an after-fix survival function ($S_C(t, Age_C)$. Such functions can be given by domain experts or learned from past data, but neither were available to us. There, we devised the following after-fix survival function $S_C(t, Age_C) = (S_C(t))^P$, where P is a parameter we call the **fix punish factor**. This after-fix survival curve holds the intuitive requirement that a replaced component is more likely to survive longer than a component that was fixed (Equation 2). The **punish-factor parameter** P **controls the difference between the after-fix and the regular survival function**. Figure 1 shows the survival curves after a punish factor of 2 and 5. Another important parameter in this set of experiments is the ratio between $C_{replace}$ and C_{fix}. We refer to this parameter as the **cost ratio** parameter. We experimented with a range of values for these two parameters and studied their impact on the long-term costs.

Figure 5 shows the results of the long-term experiments on system $S1$. The x-axis shows different cost ratios, in buckets of punish-factor values. The y-axis shows the long-term troubleshooting costs. The impact of both parameters is clear. When cost ratio is small, then Fix is significantly cheaper then Replace, and thus the Always Fix (AF) algorithm performs best. Similarly, when the punish factor is very high, a fixed component is much more likely to fail than a replaced one, thus Always Replace (AR) performs best. The Hybrid algorithm successfully chooses when to replace or fix in most parameter combinations. The same trends were also observed for system $S2$. Thus, even though the assumptions in which *DR1* (=Hybrid) is optimal do not hold in our experiments (e.g., a component may have more than two faults), we see that using it allows an effective balance between AF and AR.

7 Conclusion

We suggested the use of prognosis tools, and in particular survival curves, to lower troubleshooting costs. Two scenarios are presented where this integration of prognosis and diagnosis is useful. First, improving troubleshooting costs by using fault predictions from survival curves as priors in an MBD algorithm. Second, developing an anticipatory troubleshooter that chooses whether a faulty component should be fixed or replaced by considering possible future troubleshooting costs. Experimental results over two BN-based system models show the benefits of the developed survival-curve-aware TAs.

This works opens the possibility of future research on anticipatory troubleshooting. For example, to consider a decaying model of long-term costs instead of pre-defining a time range $[0, T_{limit}]$ in which to minimize costs.

8 Acknowledgments

This research was funded in part by ISF grant # 363/12 to Meir Kalech.

References

[Balaban and Alonso, 2012] Edward Balaban and Juan J Alonso. An approach to prognostic decision making in the aerospace domain. In *Annual Conference of the PHM Society*, 2012.

[Barto *et al.*, 1995] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[Choi *et al.*, 2011] Arthur Choi, A Darwiche, L Zheng, and Ole J Mengshoel. A tutorial on bayesian networks for system health management. *Machine Learning and Knowledge Discovery for Engineering Systems Health Management*, 10(1):1–29, 2011.

[De Kleer and Williams, 1987] Johan De Kleer and Brian C Williams. Diagnosing multiple faults. *AIJ*, 32(1), 1987.

[Elmishali *et al.*, 2016] Amir Elmishali, Roni Stern, and Meir Kalech. Data-augmented software diagnosis. In *IAAI/AAAI*, 2016.

[Eyal *et al.*, 2014] Ayelet Eyal, Lior Rokach, Meir Kalech, Ofra Amir, Rahul Chougule, Rajkumar Vaidyanathan, and Kallappa Pattada. Survival analysis of automobile components using mutually exclusive forests. *IEEE T. Systems, Man, and Cybernetics: Systems*, 44(2):246–253, 2014.

[Feldman *et al.*, 2010] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Approximate model-based diagnosis using greedy stochastic search. *JAIR*, 38:371–413, 2010.

[Feldman *et al.*, 2013] Alexander Feldman, Helena Vicente de Castro, Arjan van Gemund, and Gregory Provan. Model-based diagnostic decision-support system for satellites. In *IEEE Aerospace Conference*, pages 1–14, 2013.

[Ferreiro *et al.*, 2012] Susana Ferreiro, Aitor Arnaiz, Basilio Sierra, and Itziar Irigoien. Application of bayesian networks in prognostics for a new integrated vehicle health management concept. *Expert Systems with Applications*, 39(7):6402–6418, 2012.

[Friedrich and Nejdl, 1992] Gerhard Friedrich and Wolfgang Nejdl. Choosing observations and actions in model-based diagnosis/repair systems. *KR*, 92:489–498, 1992.

[González-Sanchez *et al.*, 2011] Alberto González-Sanchez, Rui Abreu, Hans-Gerhard Groß, and Arjan J. C. van Gemund. Spectrum-based sequential diagnosis. In *AAAI*, 2011.

[Heckerman *et al.*, 1995] David Heckerman, John S Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995.

[Ibrahim *et al.*, 2005] Joseph G Ibrahim, Ming-Hui Chen, and Debajyoti Sinha. *Bayesian survival analysis*. Wiley Online Library, 2005.

[Jannach and Schmitz, 2014] Dietmar Jannach and Thomas Schmitz. Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. *Automated Software Engineering*, 1:1–40, 2014.

[McNaught and Zagorecki, 2009] Ken R McNaught and Adam Zagorecki. Using dynamic bayesian networks for prognostic modelling to inform maintenance decision making. In *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1155–1159, 2009.

[Mengshoel *et al.*, 2010] O.J. Mengshoel, M. Chavira, K. Cascio, S. Poll, A. Darwiche, and S. Uckun. Probabilistic model-based diagnosis: An electrical power system case study. *IEEE T. on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(5):874–885, 2010.

[Miller Jr, 2011] Rupert G Miller Jr. *Survival analysis*, volume 66. John Wiley & Sons, 2011.

[Nilsson, 1982] Nils J Nilsson. *Principles of artificial intelligence*. Springer, 1982.

[Pernestål *et al.*, 2012] Anna Pernestål, Mattias Nyberg, and Håkan Warnquist. Modeling and inference for troubleshooting with interventions applied to a heavy truck auxiliary braking system. *Engineering Applications of Artificial Intelligence*, 25(4), 2012.

[Stern *et al.*, 2015] Roni Stern, Meir Kalech, Shelly Rogov, and Alexander Feldman. How many diagnoses do we need? In *AAAI*, 2015.

[Stern *et al.*, 2016] Roni Stern, Meir Kalech, and Hilla Shinitzky. Implementing troubleshooting with batch repair. In *AAAI*, 2016.

[Struss and Price, 2003] Peter Struss and Chris Price. Model-based systems in the automotive industry. *AI magazine*, 24(4), 2003.

[Tobon-Mejia *et al.*, 2012] DA Tobon-Mejia, Kamal Medjaher, and Noureddine Zerhouni. Cnc machine tool's wear diagnostic and prognostic by using dynamic bayesian networks. *Mechanical Systems and Signal Processing*, 28:167–182, 2012.

[Torta *et al.*, 2014] Gianluca Torta, Luca Anselma, and Daniele Theseider Dupré. Exploiting abstractions in cost-sensitive abductive problem solving with observations and actions. *AI Commun.*, 27(3):245–262, 2014.

[Warnquist *et al.*, 2009] Håkan Warnquist, Jonas Kvarnström, and Patrick Doherty. Planning as heuristic search for incremental fault diagnosis and repair. In *Scheduling and Planning Applications Workshop (SPARK) at ICAPS*, 2009.

[Williams and Nayak, 1996] Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *AAAI*, pages 971–978, 1996.

[Williams and Ragno, 2007] Brian C Williams and Robert J Ragno. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.

[Zamir *et al.*, 2014] Tom Zamir, Roni Stern, and Meir Kalech. Using model-based diagnosis to improve software testing. In *AAAI*, 2014.