Modeling and Troubleshooting with Interventions Applied to an Auxiliary Truck Braking System ⋆

Anna Pernestål * **Håkan Warnquist** ** **Mattias Nyberg** *

* *Dept. Electrical Engineering, Linköping University, Sweden (e-mail: {annap,matny}@isy.liu.se)*
** *Dept. Computer Science, Linköping University, Sweden (email:g-hakwa@ida.liu.se)*

Abstract: We consider computer assisted troubleshooting of complex systems, where the objective is to identify the cause of a failure and repair the system at as low expected cost as possible. Three main challenges are: the need for disassembling the system during troubleshooting, the difficulty to verify that the system is fault free, and the dependencies in between components and observations. We present a method that can return a response anytime, which allows us to obtain the best result given the available time. The work is based on a case study of an auxiliary braking system of a modern truck. We highlight practical issues related to model building and troubleshooting in a real environment.

Keywords: automobile industry; decision support systems; diagnosis; diagnostic inference; fault diagnosis; probabilistic models.

1. INTRODUCTION

Modern automotive mechatronic systems are often complex products integrating electronics, mechanics and software. Due to their intricate architecture and functionality they are often difficult to troubleshoot for a workshop mechanic. With computer aided troubleshooting the cost for troubleshooting and repair can be reduced and less experienced mechanics can be supported during their work.

Inspired by an application study of an auxiliary heavy truck breaking system, called the *retarder*, we develop a novel decision theoretic approach to troubleshooting. The objective is to find a sequence of repairs and observations that leads to a fault free truck at lowest expected cost. Earlier application studies typically consider electronic systems, such as printers and electronic control units (Heckerman et al. [1995], Langseth and Jensen [2002], Olive et al. [2003]). In comparison with these earlier application studies, the automotive mechatronic system considered here imply that the solution to the troubleshooting problem needs to take a number of additional issues into account.

First, in automotive mechatronic systems it is not as straightforward to determine whether a repair have solved the problem. In the previous works, it is assumed that after each repair it is verified whether the system is fault free or not. Such a verification is often expensive in automotive mechatronic systems, and therefore it is not presumed in the present work. This means that we need to compute probabilities after interventions, i.e. after changing the system with the repairs. Second, automotive mechatronic applications typically contains dependencies in between faults that arise during operation. These dependencies change when intervening with the system, and complicates the probability computations further, see e.g. Pearl [2000]. Third, not all parts of the retarder can be reached without first disassembling other parts of the system. This means that the level of disassembly, and the extra time required for disassembly and assembly actions, needs to be considered in the solution.

During troubleshooting the aim is to guide the mechanic by finding the next repair or observation such that the expected repair cost is minimized. Here, the troubleshooting problem is formulated as a decision-theoretic problem. The troubleshooting system consists of an action planner and a diagnoser. In the diagnoser, probabilities for combinations of faults are computed using a BN.

⋆ Supported by Scania CV AB, IVSS, and Vinnova VICT

In the planner the probabilities are used to solve a general search problem in an AND/OR graph. An optimal solution is guaranteed if sufficient computing time is allowed. Since total repair time is crucial and longer waiting times for the mechanic is generally not acceptable, the time to find the solution, i.e. the next action for the mechanic, is crucial. Therefore we emphasize on the anytime behavior of the proposed solution. That is, the proposed solution quickly computes an action leading to an acceptable repair cost and also that, for every additional computation time allowed, the expected repair cost is considerably reduced by optimizing the choice of the next action.

We begin by presenting the retarder system and discussing modeling issues in Sections 2 and 3. We then present the troubleshooting system in Section 4 before summing up with application results in Section 5.

2. THE RETARDER

The retarder is an auxiliary hydraulic braking system that allows braking of the truck without applying the conventional brakes. It consists of a mechanical system and a hydraulical system, and is controlled by an electronic control unit (ECU)The retarder generates breaking torque by letting oil flow through a rotor driven by the propeller axle causing friction. The kinetic energy is thereby converted into thermal energy in the oil that is cooled off by the truck's cooling system. At full effect and high rpm, the retarder can generate as much torque as the engine.

The retarder, which is a representative system of heavy duty trucks, is difficult to troubleshoot due to its complexity and the combination of both mechanical, hydraulical and electronical components.

3. MODELING FOR TROUBLESHOOTING

The retarder is a set of *components* which may be faulty or fault free, and which can be repaired. During troubleshooting the retarder often must be assembled or disassembled. For example to replace the oil pressure sensor, the retarder oil needs to be drained and the oil cooler needs to be removed. Each such disassemblable part is called an *assembly element*. An *action* is variable defined by its requirements on the state of the assembly elements, its cost, and its *effects*. An effect is either an observation, a repair, or a mode change of an assembly element, and generates a *request* to the mechanic.

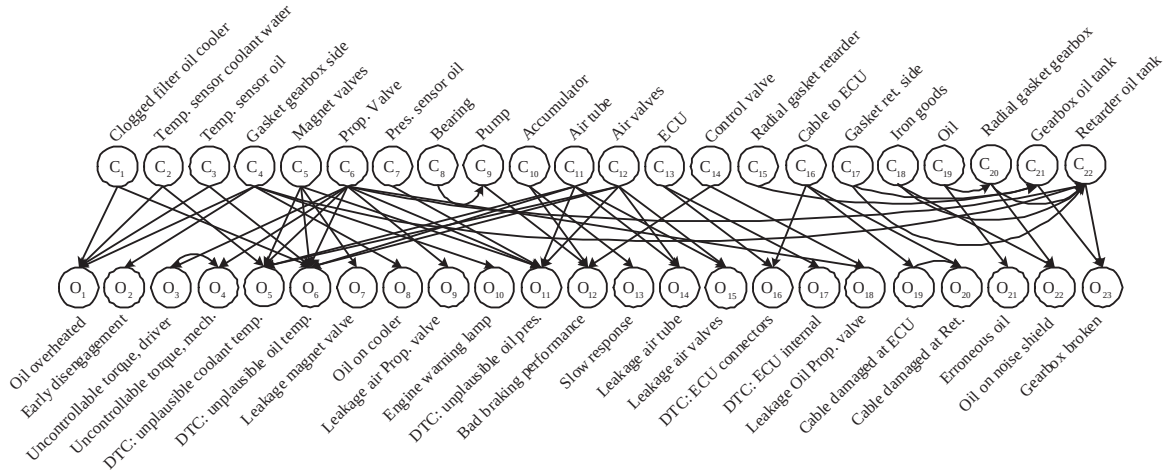

Fig. 1. A Bayesian network for the retarder

In the paper we use capital letters for variables and lower case letters for their values, e.g. $C = c$. Vectors are written in bold face. For probability distributions we write $P(c)$ to denote the probability that $C = c$. For variables in all kinds of graphs, $pa(X)$ denotes the parents of X and $ch(X)$ denotes the children of X.

In the remainder of this section we describe the different models used: a Bayesian network for probability computations, an assembly model describing the relations between assembly elements, and finally the modeling of actions.

3.1 Bayesian Network for Troubleshooting

We use a Bayesian network (BN) to model dependencies in the retarder. A BN is a directed acyclic graph where variables are represented by nodes and dependencies are represented by directed edges. See for example Jensen [2001] for a reference on BN.

A BN for troubleshooting consists of two types of variables (nodes): *components* and *observations*. Components are denoted C_i, and have the two states No Fault (NF) and Faulty (F). Observations are represented by variables O_j, and represent observations that can be made, e.g. *Air leakage at Proportional valve* and *Engine warning lamp*. Observations are typically driver's observations, observations made in the workshop, Diagnostic Trouble Codes (DTC:s) generated in the ECU during driving, or direct observations of components. A direct observation is obtained by direct inspection of a component whether it is faulty or not.

In Figure 1 a BN for the retarder *during operation* of the system is shown. The BN is based on engineers expert knowledge, and consists of 22 component nodes, denoted C_1 - C_{22}, and 23 observation nodes, denoted O_1 - O_{23}. Direct observations of components are not shown in Figure 1.

3.2 Practical Issues when Building BN for Troubleshooting

In most cases, components are parents to observations. However, there are deviations from this structure which complicates the troubleshooting task.

Components There are several ways to choose the components in the BN. The maximum size of components are sets of parts of the retarder that always are repaired together, also called *minimal repairable unit*. Choosing larger components may lead to that more parts than necessary are replaced during troubleshooting. Choosing smaller sets of parts of the retarder as components in the BN is possible, but gives worse performance in the troubleshooting algorithm and may give more parameters that need to be determined in CPT:s. Here we choose components to be minimal repairable units. Furthermore, we allow several components to be faulty at the same time.

Driver or Mechanic Observations concerning the performance of the vehicle, for example the braking torque, can be obtained by asking the driver or by letting the mechanic perform a test drive. In general, the answer from the mechanic is less uncertain but is often obtained at a higher cost since it is more expensive to let the mechanic perform a test drive than interviewing the driver. The driver's answers can only be obtained at the beginning of troubleshooting. It may be the case that the driver's answers bias the mechanic. For example, if the driver complains about uncontrollable braking torque it is reasonable that the mechanic will be influenced and observe the same symptom with higher probability. This case is modeled as a dependency between the observation nodes, see O_4 and O_3 in Figure 1 for an example.

Several Observations of Same Component Some components can be observed at several places, for example the cable between the retarder and the ECU can be observed both at the ECU, O_{19} and at the retarder O_{20}. They are two different observations since they need two different assembly states. However, there is a dependency between the two observations, since if the cable is found broken at the ECU it is less likely to be damaged at the retarder *also*.

Perception In some observations there may be uncertainties. For example the observation *Leakage air tube* (O_{14}) can be mistaken for *Leakage air valves* (O_{15}). We model this by adding dependencies from both components (tube and valve package) that can be mistaken for. We give these observations three possible values: "Sure", "Ambiguous", and "No leakage".

3.3 Modeling Observations

An observation is an indicator of faults in a subset of the components, which are modeled as parents to the observation node. There may be false indicators as well as missed faults. When an observation is performed, evidence is added to the corresponding node. The value of an observed observation is assumed to be the same until at least one of its parent components is repaired. For example if Oil on cooler (O_8) is observed, the result will be the same until the Gasket on gearbox side (C_4) is replaced. Except that this way of modeling is natural for most of our observations, it also prohibit the troubleshooting algorithm from being trapped in cycles where the same observations is made over and over again.

There may be direct dependencies between observations. We assume that all observations that are directly dependent have the same parents. This is the case for the two types of dependencies between observations described in Section 3.2.

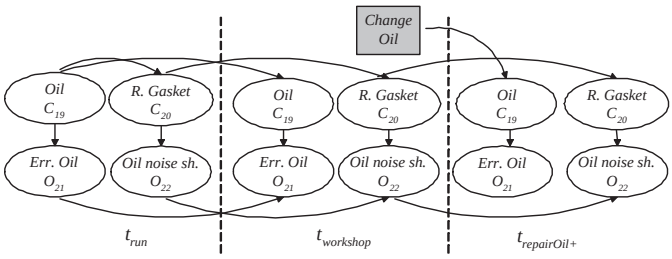

Fig. 2. Dependencies are different during operation, when arriving at the workshop, and after a repair.

3.4 Modeling Repairs

Repairs are assumed to be always successful, meaning that a component is known to become fault free after repair, and that no other faults are introduced during repair [1] . However, it is not known whether the repair action made the truck fault free, since we could have repaired an already fault free component, or, since we handle multiple faults, there may be other components that are still faulty.

The criterion for ending troubleshooting is that the posterior probability that no fault is present is large enough. Either this happens automatically during troubleshooting, or it can be verified by performing a verifying observation (typically resembling the system and perform a test run). Verification is often expensive, and it can not be assumed that verification is made after each repair.

Faults can only appear during operation of the system. During troubleshooting, the system is paused and no new faults can appear. This also means that causal dependencies between faults are different during operation and at the workshop. This, in combination with the fact that we not presume verification after repair force us to handle interventions in the BN. This is further illustrated in the following example.

Consider the causal graph in Figure 2, which describes a subpart of the retarder. In Figure 2 nodes represent observations, components, and repairs, and edges denote causal dependencies. In the retarder, a faulty *Oil* (C_{19}) may cause the *Radial Gasket at gearbox* (C_{20}) to brake during operation. When the truck arrives at the workshop, the observation *Erroneous Oil* (O_{21}) will change our opinion about in the Radial Gasket at gearbox. However, after replacing Oil (repair of C_{19}) there is no longer any dependency between the Oil and the Radial Gasket until the retarder has been run again. In Figure 2 the three different sections represent different situations. The leftmost section (denoted t_{run}) shows dependencies during operation, the middle section ($t_{workshop}$) shows dependencies when the truck has just arrived to the workshop, and the rightmost section ($t_{repairOil^+}$) shows dependencies after repair of C_{19}. In the figure, dependencies between components at different times illustrates that components do not change unless repaired. The arcs between observations mean that we can only perform an observation once, as described in Section 3.3.

After repairing a component, its related observations can be performed again. Sometimes the observations need a test drive in practice. We assume that time time scale for faults to affect observations is short in relation to the time for a component to affect other components to be faulty. Therefore, we can still use our assumption that no new faults appear during troubleshooting.

In Breese and Heckerman [1996] probability updates with interventions are handled using so called persistence networks, where mapping nodes are used to track dependency changes. In Langseth and Jensen [2002] computing probabilities with interventions are avoided, since it is assumed possible to always verify the consequence of the repair. Another attractive approach in diagnosis is to utilize Dynamic Bayesian Network (DBN), see e.g. Weber et al. [2006]. However, in our settings, interventions, the change of causality, and the fact that an observation is the same until one of its parents is repaired complicates the application of DBN.

[1] This assumption can be relaxed in the framework, but may lead to non-optimal troubleshooting

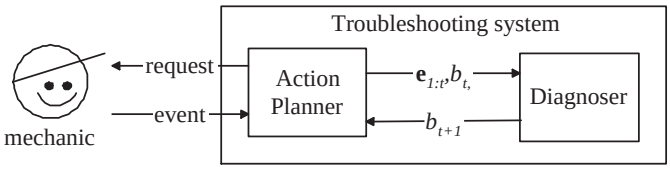

Fig. 3. The troubleshooting system.

In the current work we take a another approach, we begin with the BN describing the system during operation (or equally: just before troubleshooting begins) and update the BN as repairs are performed. With the "almost two-layer-structure" shown in Figure 1 the probability computations become simple with this approach. The details of the computations and updates are presented in Section 4.1.

3.5 Assembly Model

As mentioned in the beginning of Section 3, an *assembly element* is a disassemblable part of the vehicle such as the noise shield under the retarder or the oil cooler. Each assembly element can be in one of two modes, $assembled$ or $disassembled$. We model the relations between assembly elements as a directed acyclic graph called the *assembly graph* where each node represents an assembly element. To be in the mode $assembled$ all *children* of the node need to be in the mode $assembled$ and to be in the mode $disassembled$ all *parents* of the node needs to be in the mode $disassembled$. The *assembly state* is an assignment of modes to all assembly elements. In contrast to the state of the components, the assembly state is fully observable.

3.6 Modeling Actions

When troubleshooting the retarder, the mechanic can choose between 70 actions to perform. Each action A_i has a *base cost*, a set of *preconditions* $\mathcal{P}$, and an ordered set of *effects* $\mathcal{E}$. The preconditions are all of the type $\delta = x$ where $x \in \{assembled, disassembled\}$ and δ is an assembly element. The effects can be to repair a component C, $repair(C)$, to observe the value of an observation O in the Bayesian network, $observe(O)$, or to assemble or dissassemble an assembly element δ, $assemble(\delta)$ or $disassemble(\delta)$.

For each component C_i there is at least one action with the effect $repair(C_i)$ and for each observation O_i, in the BN, there is at least one action with the effect $observe(O_i)$. For each assembly element δ_i there is exactly one action with the effect $assemble(\delta_i)$ and exactly one action with the effect $disassemble(\delta_i)$.

For example the action Replace Oil Pressure Sensor (A_7) has base cost $cost(A_7) = 175$, preconditions $\mathcal{P}(A_7) = \langle \delta_4 = disassembled, \delta_8 = disassembled \rangle$, and effect $\mathcal{E}(A_7) = \{repair(C_7)\}$. Actions can have more than one effect, e.g. when the mechanic removes the noise shield the observation *Oil on noise shield* (O_{25}) will be made even if this was not the reason for removing the noise shield. Therefore the action *Remove noise shield* (A_{62}) is modeled with the effects $\mathcal{E}(A_{62}) = \{disassemble(\delta_2), observe(O_{25})\}$.

4. TROUBLESHOOTING SYSTEM

The troubleshooting system consists of two subsystems: the *diagnoser* and the *action planner*, see Figure 3. The actionplanner determines the next action so that the expected cost of repairing the vehicle becomes as low as possible, and the suggests the

request caused by that action to the mechanic. As described in Section 3 requests are operations to be applied to the truck, such as $repair(C)$, $observe(O)$, and $assemble(\delta)/disassemble(\delta)$. The mechanic returns the outcome of the request, The outcome of a request is called an event, and is either that C is repaired, the values $O = o$ of the observation, or that they system is assembled/disassembled.

To be able to determine the next action the planner creates a conditional plan of actions called a *troubleshooting strategy* and uses the diagnoser to predict the outcome of future actions. The diagnoser uses the BN to compute the probability distribution over possible combinations of component states given all events. The probability distribution over the component states is called the *belief state*. If an assignment of component states to all components has probability larger than zero is called a diagnosis.

4.1 Diagnoser

The planner sends the previous belief state, i.e. the probability distribution $b_{t-1} = P(\mathbf{c}^{t-1}|\mathbf{e}_{1:t-1})$ and the ordered set $\mathbf{e}_{1:t} = (e_1, \ldots, e_t)$ of events up to and including the last event to the diagnoser. The diagnoser determines the current belief state b_{t+1}. As described in Section 3.6 an action can lead to a sequence of requests, and thus a sequence of events. In the diagnoser, events are handled recursively, and it is sufficient to study the probability updates for one event at the time. We use the convention that a time step is taken after each new evidence. In accordance with e.g. Jensen [2001] we call an assignment of a variable in the BN an evidence. Events concerning observations and repairs are evidence.

General Idea The main idea is to use a Bayesian network (BN) to answer queries in the probability computations. However, as discussed in Section 3.1 repairs change dependencies in the BN and therefore the BN $\mathcal{B}_t$ must be updated as repairs are performed during troubleshooting.

Let $\mathcal{B}_t$ denote the BN at time t. When troubleshooting begins, at time $t = 1$, we initialize the BN to one describing the system during operation. For the retarder this BN is shown in Figure 1, and for the small example in Figure 2 it is represented by the leftmost section. As repairs are performed, we update the BN from $\mathcal{B}_t$ to $\mathcal{B}_{t-1}$.

Let $\mathbf{c}^t = (c_1^t, \ldots, c_N^t)$ be the component states at time t. We have that

$$P(\mathbf{c}^t|\mathbf{e}_{1:t-1}) = P(\mathbf{c}^{t-1}|\mathbf{e}_{1:t-1}), \tag{1}$$

meaning that observations and repairs made at times $1, \ldots, t-1$ do not change the states of the components from time $t-1$ to time t. To simplify notation we omit subscript t on the components when referring to the components in $\mathcal{B}_t$

Below, we explain how the belief state and then BN are updated as observation and repair events are obtained. We do not consider assemble/disassemble events since they do not affect the belief state.

Observation event First, let the event at time t be an observation $e_t = o_j$. We can then, by using (1), update the belief state b_t recursively as

$$b_t(\mathbf{c}) = P(\mathbf{c}|\mathbf{e}_{1:t-1}, o_j) = \frac{P(o_j|\mathbf{c}, \mathbf{e}_{1:t-1})b_{t-1}(\mathbf{c})}{\rho_{\mathbf{e}_{1:t}}}, \tag{2}$$

where $\rho_{\mathbf{e}_{1:t}}$ is a normalization constant independent of $\mathbf{c}^t$, and b_{t-1} is the belief state at the previous time step. To determine the third term in (2),

$$P(o_j|\mathbf{c}, \mathbf{e}_{1:t-1}), \tag{3}$$

we use the BN $\mathcal{B}_t$. Due to the "almost two-layer"-structure of the BN this probability is often easily computed. In particular, in where O_j has no direct dependencies to other components, it is simply the conditional probability for the observation O_j that can be directly found in $\mathcal{B}_t$.

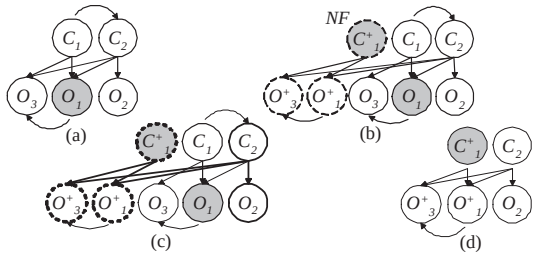

Fig. 4. Schematic picture of the change of the BN when repairing component C_1.

Repair Now, let the new event be the repair of component i. Repairing a component means that we *force* the component to be fault free by intervention, rather than *observing* it as being fault free. Therefore, it is not sufficient to only add the evidence $C_i = NF$ to the BN. The difference between interventions and observations is carefully discussed in Pearl [2000]. Using the nomenclature from Pearl [2000] we write $do(C_i = NF)$ to denote repair of component i.

After a repair we update the belief state as

$$\begin{aligned} b_t(\mathbf{c}) &= P(\mathbf{c}|\mathbf{e}_{1:t-1}, do(C_i = NF)) = \\ &= \begin{cases} 0 & \text{if } c_i = F \\ b_{t-1}(\mathbf{c}) + b_{t-1}(\hat{\mathbf{c}}) & \text{if } c_i = NF \end{cases} \\ &\text{where } \hat{\mathbf{c}} = (\hat{c}_1, \ldots, \hat{c}_N), \hat{c}_j = c_j,\ j \neq i, \text{ and } \hat{c}_i = F. \end{aligned} \tag{4}$$

This means that $\mathbf{c}$ and $\hat{\mathbf{c}}$ are the same assignments of component states except for component j, which is faulty in $\hat{\mathbf{c}}$ and fault free in $\mathbf{c}$ (by the condition in (4)).

Updating the BN When the event is an observation, dependencies between nodes in the BN are not affected and we keep the current BN, $\mathcal{B}_{t+1} = \mathcal{B}_t$.

To investigate how a repair affects the BN, study the example with two components and three observations in Figure 4. Figure 4(a) describes the system at time t and the gray node denotes that O_1 is observed. In Figure 4(b), component 1 is repaired. To represent the repair we add a new variable C_1^+ (dashed) that denotes the state of component 1 after the repair. Furthermore, new variables O_1^+ and O_3^+ (dashed) are added to represent the observations related to component 1 after the repair. The new component C_1^+ is known to work correctly, so evidence is added to this node and mark it with gray. The new observations on the other hand, have unknown values. Furthermore, since the system is paused during troubleshooting, the component C_1^+ after repair can not cause any new faults in other component, nor be affected by any other component. Thus, even if there is a direct dependency between C_1 and C_2, there is no dependency between C_1^+ and C_2.

Now note that in (3), the probabilities that are computed using the BN are always the probability of an observation o_j conditioned on the states of the components after the last repair. For the system in Figure 4, for example, we may compute the probability of any of the observations O_1^+, O_3^+, or O_2, conditioned on $\{C_1^+ = NF, C_2 = c_2\}$. After the repair of component 1, $observe(O_3)$ is a request that the planner never will ask, and thus the the probability for $O_3 = o_3$ is not of interest any more. Furthermore, probabilities of the type (3) are conditioned on the complete vector of component states. Therefore, no old observations will be of interest, unless they are direct causes of the new observations. By the assumptions on the observations in Section 3.3, only observations that have the same parents are allowed to have direct dependency relations. Thus, there is always a new copy of the observations that are children or parents of an observation of a component after repair. This leads to that there are nodes in BN 4(b) that will never be used. In Figure 4(c) nodes and dependencies that will be used in

computations after the repair of component 1 are marked with bold lines.

Since we always consider queries of the kind (3), we can safely remove the nodes that are not used in future computations. Removing these unused nodes from Figure 4(c) gives Figure 4(d).

Now, if in Figure 4(d) we rename the nodes C_1^+, O_1^+, and O_3^+, to C_1, O_1, and O_3 we obtain the BN $\mathcal{B}_{t+1}$ to be used for next evidence.

Instead of introducing new nodes for variables, deleting old nodes and updating node names as in Figure 4 we can summarize the BN updating in the following four steps taken when a repair of component i is performed.

(i) Update the belief state with the repair action $(C_i = NF)$ according to (4).
(ii) Add evidence $C_i = NF$.
(iii) Remove edges between C_i and all other components, both incoming and outgoing.
(iv) Remove evidence from all observations $O_j \in ch(c_i)$.

The update procedure presented above do not give a BN that describes the current system correctly, meaning that we can not pose arbitrary queries. However, it guarantees the correct answers to the particular queries of the type (3) that we need for our computations. Furthermore, it tracks and illustrates how dependency relations change in between components during troubleshooting.

4.2 Action Planner

The task of the action planner is to suggest the next action. To decide which action this is, the action planner searches for a *troubleshooting strategy* that, if executed to end, yields a minimal expected cost of repair given the current system state. The time spent calculating a complete troubleshooting strategy would affect the total cost of repair if the mechanic is actively waiting for a response. Therefore, if required, the action planner will terminate early and return the currently most promising partial troubleshooting strategy.

Troubleshooting Strategies A *troubleshooting strategy* π is a rooted tree in which each node n is associated with an action a_n and a system state s_n. The system state consists of the assembly state in node n, the events performed at the path to n, and the belief state in n: $s_n = \langle \mathbf{d}_n, \mathbf{e}_{1:n}, b_n \rangle$ Associated to each outgoing edge from n to a child node m in the troubleshooting strategy is a possible outcome of a_n, $u_{n,m}$, and the likelihood $l_{n,m}$ of having the outcome $u_{n,m}$ when a_n is performed in s_n. The system state of the root node corresponds to the current system state. The system state of a node m with parent node n is the resulting system state of performing a_n in s_n and having the outcome $u_{n,m}$. In a *complete troubleshooting strategy* the system state of each leaf node is a *goal state*. A goal state is a system state where the probability that the vehicle is fault free is one. The action in such a leaf node is the action that restores the vehicle to a fully assembled state. If any leaf node of a troubleshooting strategy is not a goal state, it is said to be a *partial troubleshooting strategy*.

Expected Cost of Repair The *expected cost of repair* of a troubleshooting strategy π_n rooted in a node n with system state s_n is denoted $\mathrm{ECR}(\pi_n, s_n)$. This is the expected cost of reaching any leaf node in π_n. In a node n, the probability of reaching the subtree π_m rooted in the child node m is the likelihood $l_{n,m}$. Let $cost(a_n, s_n)$ be the cost of performing a_n in s_n, then the expected cost of repair can be expressed recursively as

$$\mathrm{ECR}(\pi_n, s_n) = cost(a_n, s_n) + \sum_{m \in ch(n)} l_{n,m} \mathrm{ECR}(\pi_m, s_m).$$

Let $\Pi(s)$ be the set of all possible complete troubleshooting strategies with the system state s in the root, then the complete troubleshooting strategy π^* is an optimal troubleshooting strategy in s if

$$\pi^* = \arg\min_{\pi \in \Pi(s)} \mathrm{ECR}(\pi, s). \quad (5)$$

The expected cost of repair of π^* is the *minimal expected cost of repair*, $\mathrm{ECR}^*(s)$. This strategy can be found by, at each encountered non-goal state, choose an action a such that the expected cost of repair becomes minimal.

Proposition 1. (Minimal Expected Cost of Repair) Let n be the root node of a troubleshooting strategy with the action a_n and the system state s_n. Then the *minimal expected cost of repair* in s_n is

$$\mathrm{ECR}^*(s_n) = \min_{a_n}(cost(a_n, s_n) + \sum_{m \in ch(n)} l_{n,m} \mathrm{ECR}^*(s_m)).$$

Applicable Actions Not all actions need to be considered when deciding candidates to be included in the optimal troubleshooting strategy. We only need to consider actions that can affect the belief state part of the system state. These actions are *applicable actions*. Applicable actions in a system state must be actions that repair faults with a marginalized probability greater than zero or makes observations that are causally dependent on such a fault.

Composite Actions The preconditions are not considered when finding applicable actions. This is not needed since as stated in Section 3.6 there exists exactly one action that assembles or disassembles each assembly element. This means that there is a unique way to fulfill all preconditions. A *composite action* is created by combining actions that fulfill the non-fulfilled preconditions of the original applicable action. The cost, preconditions, and effects of these actions are added to the cost, preconditions and effects of the original action. This allows us to ignore all preconditions and focus on the desired effects without losing optimality.

Search Graph All possible choices of actions can be represented as an AND/OR graph with alternating layers of OR nodes and AND nodes . The OR nodes are labeled with system states and correspond to decision points where different actions can be chosen. The AND nodes correspond to chance nodes where the outcomes of the last action will decide the next OR node. Each different choice successing AND node to the OR nodes is a *solution* to the AND/OR graph. If the leaf nodes in a solution are all goal states the solution is *complete*, otherwise it is *partial*. There is a one-to-one correspondence between a solution and a troubleshooting strategy (Vomlelová and Vomlel [2000]), so a complete solution correspond to a complete troubleshooting strategy and partial solution correspond to a partial troubleshooting strategies.

The size of the AND/OR graph is highly exponential, but by using heuristic search algorithms such as AO^* (Nilsson [1980]), not the entire graph needs to be explored to find an optimal solution.

Since observations are modeled such that they cannot be repeated and repairs always are successful, the search graph is acyclic when only applicable actions are considered. If we wish to relax any of these assumptions the search graph may become cyclic. However, there are variants of the AO^* algorithm such as the CFC_{rev} algorithm that can treat cyclic graphs (Jiménez and Torras [2000]).

Algorithm The main parts of the AO^* algorithm are shown in Table 1. It starts out with a search graph and a partial solution consisting only of the root OR node. Until the root node is marked solved, an unsolved leaf node in the partial solution is chosen by `findUnsolvedLeaf` and expanded by `expandNode`. When expanding this node, a succeeding AND node is created for every applicable action each with succeeding OR nodes for each possible outcome of these actions. Starting from the expanded node and backtracking toward the root, the currently best solution is revised in `reviseSolution`.

```
while root is unsolved do
   nextNode := findUnsolvedLeaf;
   expandNode(nextNode);
   reviseSolution(nextNode);
end while
```

Table 1. The AO^* algorithm

A node is marked solved if all succeeding nodes are solved. The nodes in the solution are assigned costs in accordance with Proposition 1 where unsolved leafs receive an estimated cost given by a heuristic function h. As soon as the root node becomes solved we have a complete solution. This solution is optimal if the heuristic function is *admissible*, i.e. for a node n labeled with the system state s_n, $h(n) \leq \mathrm{ECR}^*(s_n)$ (Nilsson [1980]). In the current work the heuristics presented in Warnquist and Nyberg [2008] are used to find optimal and suboptimal solutions are used.

Anytime Properties Finding optimal troubleshooting for a problem as large as the model of the retarder can be very time consuming. Whenever desired by the user, the search can be aborted and the currently best partial solution is returned. When this happens, the algorithm stops expanding nodes and sets the costs in the unsolved leafs to a upper bound and revises the solution.

5. APPLICATION

The troubleshooting system described above is implemented and applied to the problem of repairing a heavy truck with a faulty retarder.

In the implementation, the diagnoser is set to disregard diagnoses where four or more components are faulty. This is done to keep the size of the belief state manageable and is reasonable since the probability for several simultaneous faults in the retarder is typically very small. In the action planner the size of the belief state is further reduced by only keeping the k most probable diagnoses. This method of keeping down the size of the belief state works for our model of the retarder, but it is not feasible for larger systems. In those cases methods as the one presented in Lerner et al. [2000] can be used, where the diagnoser collapses similar diagnoses into one.

To test the troubleshooting system we inject faults in the model of the retarder and simulate the troubleshooting process. The time required to find an optimal solution varies greatly depending on the initial observations generated by the fault. To avoid long waiting times the user can abort the search and perform a suboptimal action instead. For a randomly generated test the optimal the optimal expected cost of repair and the ECR when aborted at different times are measured. Figure 5 shows the average of these costs. Finding a solution that is guaranteed to be optimal solution requires 620 seconds using a Java implementation on a PC, but when aborted convergence is reached after 60 seconds.

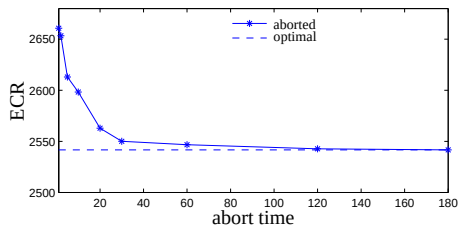

Fig. 5. The anytime solution at different abort times compared to the optimal solution.

6. CONCLUSION

Inspired by the application study of the retarder, a heavy truck breaking system, we have developed a decision theoretic approach to troubleshooting. Focus has been on issues important in real world applications: the need for disassembling the system during troubleshooting, the problem of verifying that the system is fault free, and the fact that there are dependencies in between observations and in between components. To meet the crucial requirement on short waiting times for the mechanic we have proposed a solution with anytime behavior. The solution utilizes the time available to return a best possible troubleshooting strategy, and converges toward the optimal solution as more time is available. We have applied the proposed troubleshooting approach to the retarder, and discussed carefully how to model the system and how the troubleshooting is performed.

There are still several challenging and interesting open questions. The dependencies in between components and in between faults result in complicated BN structures. The BN used here is still fairly simple, and in our future work we will investigate how interventions can be modeled in even more general BNs. The results from simulations with noisy parameters show that parameters may deviate a little from their nominal values, but in our future work we will also ask us whether deviations in certain parameters have larger impact on the result than others. Other interesting open questions are how to determine parameters in the BN. Furthermore, one challenge is the dimension of the belief state, which increases exponential with the number of components. We are currently working on methods for focusing on the most probable diagnoses in the diagnoser, without risking to loose diagnoses with small probabilities in the first time steps.

The results presented are promising, and show that computer aided troubleshooting can be applied to complex mechatronic systems such as the retarder. We look forward to extend our algorithm to troubleshoot even larger systems.

REFERENCES

John S. Breese and David Heckerman. Decision-theoretic troubleshooting: A framework for repair and experiment. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, 1996.

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

Finn V. Jensen. *Bayesian Networks*. Springer-Verlag, New York, 2001.

P. Jiménez and C. Torras. An efficient algorithm for searching implicit and/or graphs with cycles. *Artificial Intelligence*, 124(1):1–30, 2000.

Helge Langseth and Finn V. Jensen. Decision theoretic troubleshooting of coherent systems. *Reliability Engineering & System Safety*, 80(1):49–62, 2002.

Uri Lerner, Ronald Parr, Daphne Koller, and Gautam Biswas. Bayesian Fault Detection and Diagnosis in Dynamic Systems. In *AAAI/IAAI*, pages 531–537, 2000.

Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA, 1980.

Xavier Olive, Louise Trave-Massuyes, and Hervé Poulard. AO* variant methods for automatic generation of near-optimal diagnosis trees. In *14th International Workshop on Principles of Diagnosis (DX'03)*, pages 169–174. 2003.

Judea Pearl. *Causality*. Cambridge, 2000.

Marta Vomlelová and Jiří Vomlel. Troubleshooting: NP-hardness and solution methods. In *Proceedings of the Fifth Workshop on Uncertainty Processing, WUPES'2000*. 2000.

Håkan Warnquist and Mattias Nyberg. A heuristic for near-optimal troubleshooting using ao^*. In *Proceedings of the 19th International Workshop on Principles of Diagnosis*, 2008.

Philippe Weber, Didier Theilliol, Christophe Aubrun, and Alexandre Evsukoff. Increasing Effectiveness of Model-based Fault Diagnosis: a Dynamic Bayesian Network Design for Decision Making. In *Proceedings of 6th IFAC Symposium on Fault Detection, Supervision and Safety of technical processes*, pages 109–114, 2006.