Preprints of the 7th IFAC Symposium on
Fault Detection, Supervision and Safety of Technical Processes
Barcelona, Spain, June 30 - July 3, 2009

Anytime Near-Optimal Troubleshooting Applied to an Auxiliary Truck Braking System

Håkan Warnquist * Anna Pernestål ** Mattias Nyberg **

* *Dept. Computer Science, Linköping University, Sweden*
** *Dept. Electrical Engineering, Linköping University, Sweden*

Abstract: We consider computer assisted troubleshooting of complex systems, for example of a vehicle at a workshop. The objective is to identify the cause of a failure and repair a 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; heuristic searches; 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 time for troubleshooting and repair can be reduced and more inexperienced mechanics can be supported during their work.

The work is based on a case study of an auxiliary braking system of a modern truck, called the *retarder*. We develop a decision theoretic approach to troubleshooting where the objective of troubleshooting 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 mechatronic system considered here imply that the solution to the troubleshooting problem needs to take a number of additional issues into account.

Firstly, not all parts of the retarder can be reached without first disassembling other parts of the truck or retarder. This means that the level of disassembly, and the extra time required for disassembly and assembly actions, need to be considered in the solution. Secondly, in automotive mechatronic systems it is not as straightforward to determine whether the last repair has removed all faults or not. In the previous works, it is assumed that after each repair it is verified whether the system is fault free or not. This is often not possible in mechatronic systems, and such an assumption is therefore not made in the present work. Third, mechatronic applications typically contains dependencies in between faults and in between observations.

During the troubleshooting the aim is to guide the mechanic by, in each step, finding the next repair or observation, such that the expected repair cost is minimized. The approach taken here is to formulate the problem as a general search problem in an AND/OR graph. Thereby an optimal solution is guaranteed if sufficient computing time is allowed. Since total repair time includes computation time, 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. In previous work such as Langseth and Jensen [2002] and Sun and Weld [1993] computation time is kept low by using lookahead search, but that approach would not gain from situations where more computation time is available. Therefore we emphasize the anytime behavior of the proposed solution. That is, the best possible solution is calculated given the available computation time, and for every additional computation time allowed, a better solution is obtained. In the paper, we also highlight practical issues such as modeling the system adequately and investigate the sensitivity of errors in the model.

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 SYSTEM

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 cooling system of the truck. 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 THE RETARDER

The retarder is a set of *components*. Each component has two states: fault free or faulty. A component can be repaired by applying a *repair* to that component. 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

⋆ All authors are affiliated with Scania CV AB.

disassemblable part of the truck is called an *assembly element*. An *action* is 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 modification of an assembly element.

In the remainder of this section we describe the notation used and the different models used: a Bayesian network (BN) to model dependency relations between observations and components, an assembly model describing the relations between assembly elements, and finally the modeling of actions.

3.1 Notation

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 $C = c$.

3.2 Bayesian Network for the Retarder

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. The retarder BN consists of two kinds of variables (nodes): *observations* and *components*. Dependencies between nodes are modeled using directed edges. Component variables represent subsystems of the retarder that can be repaired or replaced, e.g. the Oil pressure sensor, the Oil pump, and the ECU. Observation variables represent observations that can be made, e.g. air leakage at Proportional valve, slow activation of retarder, engine warning lamp. Observations are typically Diagnostic Trouble Codes (DTC:s) generated in the ECU during driving, driver's observations, observations made in workshop, and direct observations of components. A direct observation means that it is decided by direct inspectation of a component whether it is faulty or not.

To model a system, there are several different BN:s that can be used. We use a BN where edges between variables are chosen to represent causal dependencies. This approach gives an easy interpretation of the resulting BN and facilitates local approaches when the system is updated (Pearl [2000]).

The BN describing the retarder is based on engineers expert knowledge and is shown in Figure 1. In the network there are 22 components, denoted C_1 - C_{22}, and 23 observations, denoted O_1 - O_{23}. Direct observations of components are not shown in Figure 1.

For each node a Conditional Probability Table (CPT) is needed. CPT:s for components are assigned by experts and by using manufacturer's specifications. Using full CPT:s for all nodes would require an infeasible number of parameters, already for small systems. To minimize the number of parameters needed, we use three different types of observation nodes. Firstly, most observation nodes are assumed to be *noisy-or* nodes (Jensen [2001]). This assumption keeps the number of parameters down. Secondly, direct observations of components are assigned logical CPT:s containing zeros and ones only, meaning that faulty components are detected with certainty. Finally, when special characteristics must be expressed a full CPT is used. This possibility requires many parameters, but gives full freedom, and is used for three observations in the retarder.

3.3 Practical Issues when Building BN

In most cases, components are parents to observations, but there are deviations from this structure. In the remainder of this section we discuss practical issues when building a BN for troubleshooting.

Driver or Mechanic Observations concerning the performance of the vehicle, for example the braking torque, can be obtained by asking the driver or by performing a test drive. In general, the answer from the mechanic can be assumed to be less uncertain but it is obtained at a lower cost since it is more expensive to let the mechanic perform a test drive than interviewing the driver. On the other hand, the driver's answers can only be obtained at the first time step. Furthermore, it may be the case that the driver's answers bias the mechanic. For example, if the driver complains about uncontrollable braking torque it may be reasonable that the mechanic will observe this with higher probability. This case is modeled as a dependency between the observation nodes, see O_3 and O_4 in Figure 1 for an example.

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 this gives worse performance in the troubleshooting algorithm and leads to that more parameters need to be set in CPT:s.

Here we choose components to be minimal repairable units. It may be the case that several components are faulty at the same time. Components can be repaired alone or together with another components.

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". An alternative is to add an extra layer to model the perception explicitly, but we choose the first alternative to keep the number of nodes in the BN as small as possible.

Repairs We assume that a repair is always successful, meaning that the repaired component is known to be fault free and no other faults are introduced during repair. However, it is not known whether the repair action made the truck fault free before a verifying observation has been performed.

When a repair is performed, evidence is added in the BN that the component is fault free. Furthermore, all direct edges between the repaired component and other component are removed. The reason is that dependencies between components arise during driving, for example erroneous Oil (C_{19}) may cause the Radial gasket at the gearbox (C_{20}) to break during driving. After changing Oil at the workshop, there is no dependency between the oil and the gasket.

3.4 Observations

When an observation is performed, evidence is added to the corresponding node. If the observation is repeated before at least one of its parent components is repaired the result will be the same, for example if Oil on cooler (O_8) is observed, the observation will be the same until the Gasket on gearbox side (C_4) is replaced. Except that this way of modeling observations is the most 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.

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

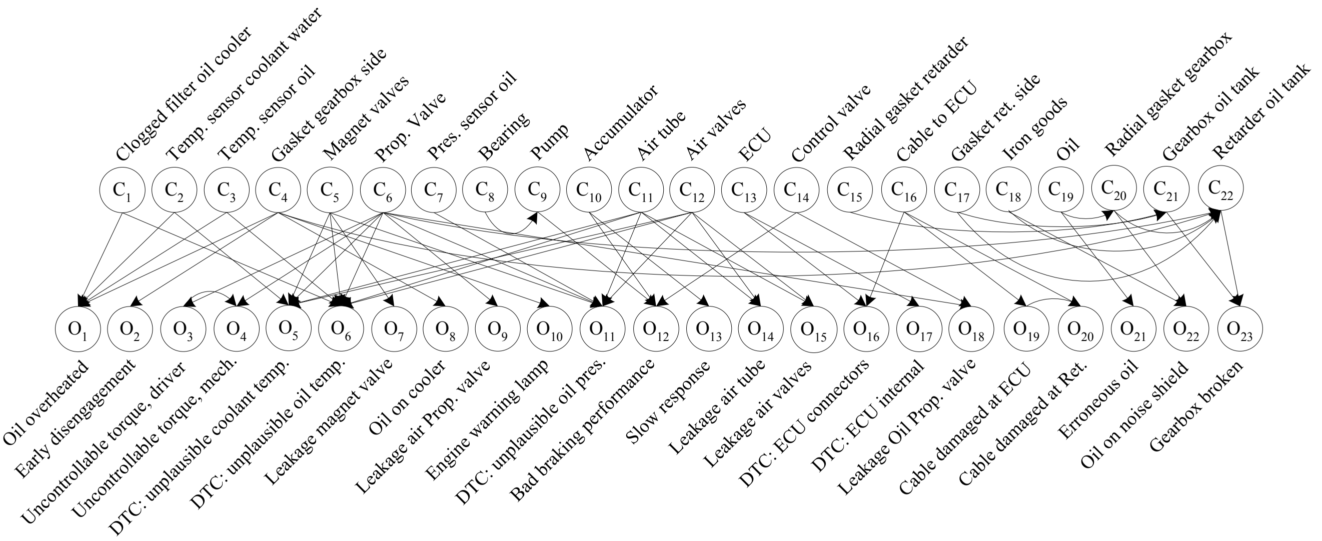

Fig. 1. A Bayesian network for the retarder

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. The assembly graph of the retarder is shown in Figure 2.

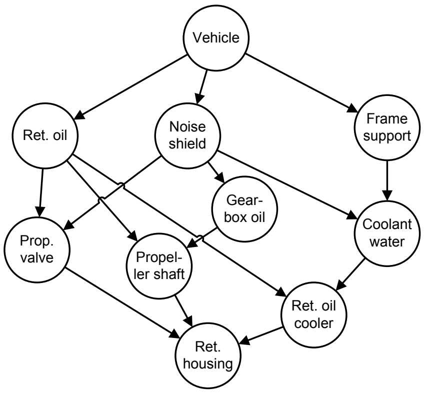

Fig. 2. The assembly graph of the retarder.

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 $disassemble(\delta)$. The base cost is a constant value corresponding to the time needed and the cost of the materials consumed when performing the action.

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 the base cost $cost(A_7) = 175$, the preconditions $\mathcal{P}(A_7) = \{\delta_4 = disassembled, \delta_8 = disassembled\}$, and the 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_{22}) 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 $\mathcal{E}(A_{62}) = \{disassemble(\delta_2), observe(O_{22})\}$.

4. TROUBLESHOOTING SYSTEM

The troubleshooting system consists of two subsystems: the *diagnoser* and the *action planner*. The planner suggests the next action to be performed so that the expected cost of repairing the vehicle is as low as possible. To be able to suggest an 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 a set of evidence. The probability distribution over the component states is called the *belief state*, and one such assignment with probability larger than zero is called a diagnosis.

4.1 Diagnoser

The planner asks the diagnoser about the belief state b_{t+1}, given the current *system state* and an action. The system state s_t at time t consists of the current assembly state $\mathbf{d}_t$, an ordered set $\mathbf{e}_{1:t}$ of repairs and observations made so far, and the current belief state b_t: $s_t = \langle \mathbf{d}_t, \mathbf{e}_{1:t}, b_t \rangle$. We use the term evidence to denote the knowledge that a certain repair or observation that is made. One action can lead to a sequence of evidence. In the diagnoser, evidences are handled recursively, and therefore it is sufficient to consider one evidence at the time.

Let $\mathbf{e}_t$ be the evidence at time t, and let $\mathbf{c}^t = (\mathbf{c}_1^t, \mathbf{c}_2^t, \ldots)$ be the component state at time t. We have that

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

meaning that observations and repairs made at times $1, \ldots, t-1$ do not change the status of the components from time $t-1$ to

t. When $e_t = o_j$, i.e. when evidence is an observation o_j , we update the belief state b_t according to

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

where $\rho_{\mathbf{e}_{1:t}}$ is a constant independent of $\mathbf{c}^t$.

Let PA_i be the set of observations that are parents to node O_j and at the same time in the evidence. Since our BN for the retarder only have causal dependencies we have

$$P(o_j|\mathbf{c}^{t-1}, \mathbf{e}_{1:t-1}) = P(o^j|\mathbf{c}^{t-1}, pa_i), \tag{2}$$

which can be computed using the BN. However, most observations in our BN have no other observations as parents. In this case (2) becomes $P(o_j|\mathbf{c}^{t-1}, \mathbf{e}_{1:t-1}) = P(o_j|\mathbf{c}^{t-1})$, and we can compute (1) by simply looking up the CPT:s for the given observation.

When the evidence is a repair of component i, i.e. when $e_t = a_i$, we obtain after marginalization and some algebra the updating rule

$$\begin{aligned} &b_t(\mathbf{c}^t) = P(\mathbf{c}^t|\mathbf{e}_{1:t-1}, a_i) = \\ &= \begin{cases} P(\mathbf{c}^t_{i=0}|\mathbf{e}_{1:t-1}, a_i) + P(\mathbf{c}^t_{i=1}|\mathbf{e}_{1:t-1}, a_i) & \text{if } c_t^j = 0 \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{3}$$

where $\mathbf{c}^t_{i=x} = (c^t_1, \ldots, c^t_{i-1}, x, c^t_{i+1}, \ldots, c^t_N)$, i.e. a component state where the ith component has value x.

Probability computations for troubleshooting in more complex systems is discussed for example in Pernestål et al. [2009].

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. Associated to each outgoing edge from n to a child node m is a possible outcome of a_n, $o_{n,m}$, and the likelihood $l_{n,m}$ of having the outcome $o_{n,m}$ 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 $o_{n,m}$. 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 the 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 and let $ch(n)$ be the set of child nodes to n, then the expected cost of repair can be expressed recursively as

$$\mathrm{ECR}(\pi_n, s_n) = cost(a_n, s_n) + \sum_{m \in ch(n)} l_{n,m}\mathrm{ECR}(\pi_m, s_m). \tag{4}$$

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

$$\pi^* = \operatorname*{arg\,min}_{\pi \in \Pi(s)} \mathrm{ECR}(\pi, s). \tag{5}$$

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

$$\mathrm{ECR}^*(s_n) = \min_{a_n}\left(cost(a_n, s_n) + \sum_{m \in ch(n)} l_{n,m}\mathrm{ECR}^*(s_m)\right) \tag{6}$$

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 (see Figure 3). Each possible choice of succeeding AND nodes from the OR nodes is a different *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]). A complete solution corresponds to a complete troubleshooting strategy and a partial solution corresponds to a partial troubleshooting strategy.

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 if 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

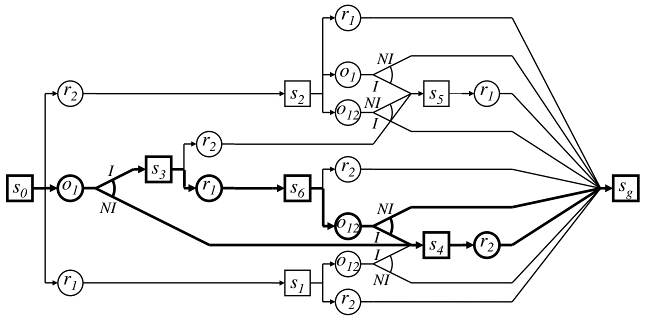

Fig. 3. Example of an AND/OR graph. Square nodes are OR nodes and circular nodes are AND nodes. One troubleshooting strategy is highlighted, describing a plan to reach a goal state s_g from the initial system state s_0. In the AND nodes, r_i are repairs and o_i are observations.

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

Table 1. The AO^* algorithm

as the LAO^* algorithm that can treat cyclic graphs (Hansen and Zilberstein [2001]).

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`. 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, $h(n) \leq \text{ECR}^*(s_n)$. (Nilsson [1980])

Heuristics The admissible heuristic function used to evaluate the cost of unsolved OR nodes is derived from a relaxation of the problem where the true diagnosis is assumed to be found at zero cost. This is the lower bound $\underline{h}$. Let $b_n \in s_n$ be the belief state in the system state s_n labeling an OR node n and let $\mathbf{c}$ be an assignment of component states with non-zero probability, i.e. a diagnosis. Furthermore, let $a_{\mathbf{c}}$ be a composite action that repairs all faults in the diagnosis and restores the system to fully assembled state having the cost $cost(a_{\mathbf{c}})$. Then the lower bound is calculated as

$$\underline{h}(n) = \sum_{\mathbf{c} \in b_n \in s_n} b(\mathbf{c}) cost(a_{\mathbf{c}}). \tag{7}$$

The closer $\underline{h}(n)$ is to $\text{ECR}^*(s_n)$ the smaller part of the entire AND/OR graph needs to be explored.

To further reduce the search graph we introduce another heuristic function, the upper bound $\overline{h}$, based on work by Heckerman et al. [1995] and Langseth and Jensen [2002]. The upper bound is the expected cost of repair using a fixed troubleshooting strategy that observe each component directly in a specific order and repairs those that are faulty. After each repair a *function control* is made. In the retarder model there is no single observation that corresponds to a function control. Instead it is approximated by a test drive with all observations available. Note however, that the function control is not performed in practice, but only used in the computations of the upper bound. If the function control indicates that further components need repair, the system is taken to the same assembly state as after the last direct observation of a component and the next component is observed. Let c_{o_i} be the cost of observing the *i*th component, let c_{r_i} be the cost of repairing the *i*th component followed by a function control, and let c_{d_i} be the cost of taking the system to from the assembly state obtained after the function control to the one obtained after observation of component *i*. If a component cannot be observed directly, c_{o_i} is the cost of repairing and controlling and $c_{r_i} = 0$. The upper bound is given by

$$\overline{h}(n) = \sum_{i=1}^{N} \left(p_{o_i} \cdot c_{o_i} + p_{r_i} \cdot c_{r_i} + p_{d_i} \cdot c_{d_i} \right) \tag{8}$$

where the probabilities for each action can be calculated from the belief state in the current system state as

$$p_{o_i} = P\Big(\bigvee_{j=i}^{N} C_j = F \Big) \tag{9}$$

$$p_{r_i} = P\big(C_i = F\big) \tag{10}$$

$$p_{d_i} = P\Big(C_i = F \wedge \Big(\bigvee_{j=i+1}^{N} C_j = F \Big)\Big) \tag{11}$$

and the components are ordered ascending by the ratio c_{o_i}/p_{o_i}.

For each node in the search graph, values for the upper bound and lower bound are stored. If the lower bound of an AND node is higher than the upper bound of another AND node sharing the same parent, this node and the entire search branch below can be pruned to save memory. In the original version of AO^* the next unsolved leaf is chosen arbitrarily. In our domain we have experienced significant speedups if the unsolved leaf m is chosen such that the value $l_{n,m} \cdot (\overline{h}(m) - \underline{h}(m))$ is the greatest where $l_{n,m}$ is the product of all likelihoods on the path from the root node n to the leaf m.

Anytime Properties Since conditional planning under partial observability is 2-EXP complete Rintanen [2004], it can be very time consuming to find optimal troubleshooting strategies for the retarder. Therefore, 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 the upper bound and revises the solution.

5. APPLICATION

We have implemented the troubleshooting system described above and applied it 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 since the probability for several simultaneous faults in the retarder is typically very small. When the current system state is passed on to the action planner the size of the belief state is reduced further 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 this case 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 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.

To illustrate how different waiting times affect the quality of the solution, we let the action planner create troubleshooting strategies for a representative test case where an optimal solution is known. Plotted in Fig. 4 is the cost of the optimal troubleshooting strategy (dotted line) and the costs of the partial troubleshooting strategies aborted at different times (solid line). Finding the optimal solution required 75 seconds using a prototype implementation on a PC, but when aborted convergence is reached after 30 seconds.

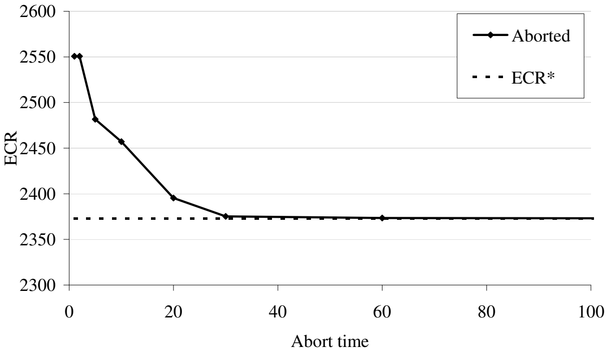

Fig. 4. Plot showing how the anytime solution at different abort times converge toward the optimal solution.

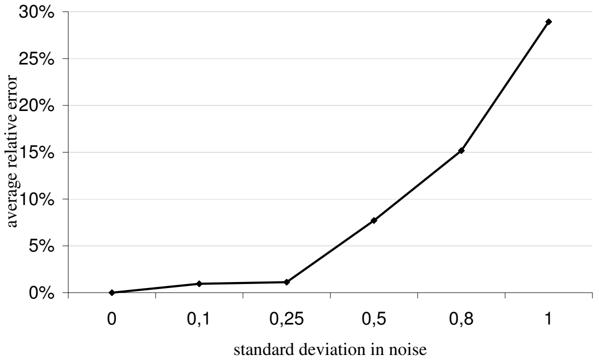

Fig. 5. Average error in ECR when using a distorted BN compared to the nominal BN in the diagnoser.

As discussed in Section 3.2, parameters in the BN must be assigned. Here, this assumed to be done by experts. To investigate the effect of the accuracy of the parameters, noise is introduced in the parameters in the BN. Noise was added using the log-odds normal distribution as described in Kipersztok and Wang [2001]. In Figure 5 the average discrepancy in the cost for troubleshooting using the noisy BN compared to using the nominal BN is plotted. Small errors in the parameters does not affect the result significantly, but for noise with standard deviation above 0.25 the error increases fast.

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 observations result in a more complicated BN than the two-layer BN that is the model used in many previous work on troubleshooting. The BN presented here is still fairly simple, and in our future work we will investigate how interventions can be modeled in even more complex BN:s.

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.

ACKNOWLEDGEMENTS

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

REFERENCES

Eric A. Hansen and Shlomo Zilberstein. LAO * : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

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.

Oscar Kipersztok and Haiqin Wang. Another look at sensitivity of bayesian networks to imprecise probabilities. In *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, 2001.

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.

Anna Pernestål, Håkan Warnquist, and Mattias Nyberg. Modeling and troubleshooting with interventions applied to an auxiliary truck braking system. In *Proceedings of 2nd IFAC workshop on Dependable Control of Discrete Systems*, 2009.

Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 345–354. AAAI Press, 2004.

Ying Sun and Daniel S. Weld. A framework for model-based repair. In *In Proc. AAAI-93*, pages 182–187, 1993.

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