Troubleshooting when Action Costs are Dependent with Application to a Truck Engine

Håkan WARNQUIST [a], Mattias NYBERG [b] and Petter SÄBY [a]
[a] *Service Methods Framework Development, Scania, SE-151 87 Södertälje, Sweden*
[b] *Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden*

Abstract. We propose a troubleshooting algorithm that can troubleshoot systems with dependent action costs. When actions are performed they may change the way the system is decomposed and affect the cost of future actions. We present a way to model this by extending the traditional troubleshooting model with an additional state that describes which parts of the system that are decomposed. The proposed troubleshooting algorithm searches an AND/OR graph with the aim of finding the repair plan that minimizes the expected cost of repair. We present the heuristics needed to speed up the search and make it competitive with other troubleshooting algorithms. Finally, the performance of the algorithm is evaluated on a probabilistic model of a fuel injection system of a truck. We show that the expected cost of repair can be reduced when compared with an algorithm from previous literature.

Keywords. Fault Diagnosis, Decision Theoretic Troubleshooting

Introduction

In many troubleshooting algorithms, where the aim is to minimize the expected cost of repair, action costs are assumed to be independent [1][2]. For complex systems such as modern trucks it can be unrealistic to make this assumption. When troubleshooting a truck, the mechanic sometimes must decompose large parts of the truck to find the faulty component in need for repair, i.e. the cab needs to be tilted and components need to be decomposed. This can be very time consuming. Depending on which actions that have previously been performed, the cost of performing other actions may vary. For trucks it is often the case that the vehicle has to be fully reassembled to test if everything is functioning properly. If the wrong repair is made the vehicle has to be decomposed again. Therefore, when mechanics repair trucks they sometimes repair more components than necessary to avoid spending too much time decomposing and assembling the vehicle.

The problem of troubleshooting with dependent action costs is addressed in [3]. There, actions are grouped in clusters such that when an action is performed within a cluster the other actions in the same cluster become cheaper. One limitation of this method is that the clusters are not allowed to be nested. Instead, in this paper we propose to model the action costs to be dependent on which parts of the system that are decomposed. This allows a more flexible description of the action costs. We propose a

troubleshooting algorithm that can troubleshoot systems with this type of dependent action costs. In contrast to some other troubleshooting algorithms [1][2][4] it may choose to repair more components than necessary before reassembling, if this can reduce the total expected cost of repair. The algorithm is evaluated on a probabilistic model of the fuel injection system of a truck. On this same model the performance of the algorithm is compared with an implementation of the decision theoretic troubleshooting algorithm proposed by Breese and Heckerman [1]. The result of the comparison shows that the here proposed algorithm can reduce the expected cost of repair significantly. The algorithm is based on an AND/OR search [5] which easily becomes computationally intractable, but with the right heuristics the search can be made in reasonable time.

The outline of the paper is as follows. In Section 1 we describe the troubleshooting model and how the dependent action costs are modeled. Section 2 describes the search algorithm and the heuristics that are used. In Section 3 the performance of the algorithm is evaluated on a model of the fuel injection system of a truck. Also, the performance of the algorithm is compared with the algorithm proposed in [6]. Finally, we conclude in Section 4.

1. The Troubleshooting Model

The troubleshooting model consists of two separated states, the *decomposition state*, describing which parts of the system that are accessible, and the *fault state*, describing which component that is faulty. The decomposition state is completely observable while the fault state is not directly observable. In this section we describe the different states of the model and the actions that can be performed.

1.1. Fault State and Belief State

The state describing which component of the modeled system that is faulty, is called the *fault state*. This state is not directly observable, otherwise there would be no need for troubleshooting. Instead, the probability of being in a certain fault state is estimated from a probabilistic model of the system using the current information at hand, such as observations and earlier performed actions, as done in [6] and [2]. The probability distribution of the fault states represents our belief in which components are faulty. Such a distribution is commonly called a *belief state* [5].

1.2. Actions

The fault state is changed by performing actions on the system. When an action is performed a new belief state can be inferred from the previous belief state and from the new information that may have been gained from the action. This type of inference is also made in [6] and [2]. We distinguish between actions that gain information, *testing actions*, and actions that revokes faults, *fault revoking actions*. The testing actions and the fault revoking actions correspond to the *questions* and the *repair actions* in [2] respectively. When a testing action is performed we retrieve a new belief state for every possible outcome of the test. When a fault revoking action is performed a single new belief state can be calculated simply by moving probability mass from fault states where the revoked fault is faulty onto fault states where it is not.

For a given belief state not all actions are relevant. For example, if we are sure that the batteries are working properly we do not need to consider fault revoking actions such as "replace batteries" or testing actions such as "measure fluid level in batteries." Relevant actions that we allow to be considered, are called *applicable actions*.

Definition 1 (Applicable Action)**.** An action a is an *applicable action* if after it is performed, the resulting belief state, b_{after}, is different from the prior belief state, b_{before}, i.e. $b_{after} \neq b_{before}$.

1.3. Modeling the Decomposition State

Consider a bicycle. Before the inner tube on the wheel of a bicycle can be changed, the wheel has to be dismounted and the tire has to be taken off. The time required for dismounting the wheel and removing the tire can be considered affecting the cost of the action *change inner tube*. A dismounted wheel is not considered to be a fault, just a normal step of the repair process, and the mechanic will always know if it is mounted or not, so there is no need to represent this fact in the belief state.

To represent parts of the system that can be taken apart during a normal repair process we introduce the *decomposition state*. This state is fully observable. The parts of the system that can be taken apart are called *decomposable units*. The notion of a decomposition state was introduced in a series of master theses [7][8][9] with the aim of troubleshooting trucks using theory from [1][6][2].

Definition 2 (Decomposition State)**.** The *decomposition state* $\delta = [d_1, d_1, \ldots]$ is a vector, where each element d_i represents a *decomposable unit* of the system. Each element is in one of the modes *decomposed* or *assembled*.

Each action may require one or more of the decomposable units to be in a certain mode. Further, decomposing a decomposable unit may require other decomposable units to be in the mode *decomposed*. This relation between the elements of a decomposition state can be described by a directed acyclic graph, see Figure 1. Each node represents a decomposable unit and requires its parents to be in the mode *decomposed*, before itself can be decomposed. Before a node can be assembled, all its children have to be in the mode *assembled*.

Changing the mode of a decomposable unit is associated with a certain cost. When actions are performed, the cost of making the transition necessary to meet the requirements of the action is added to the cost of performing the action. Note that the transitions in the decomposition state are not treated as actions themselves. A transition in the decomposition state occur only when it is required by an action that is performed. This makes the cost of performing an action dependent of previously performed actions.

Example. Consider a decomposition state where all decomposable units in Figure 1 are assembled. The cost of an action that requires d_7 to be decomposed is increased with the cost of decomposing d_1, d_2, d_4, d_5, and d_7. Afterward we wish to perform an action that requires d_1 to be assembled. The cost of this transition in the decomposition state is the cost of assembling d_1, d_4, and d_7.

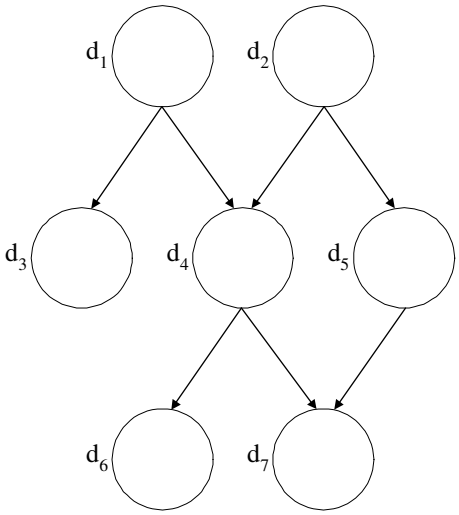

Figure 1. The relation between elements in the decomposition state. Each node, d_i, represents an individual decomposable unit. The top layer represents the outer, easily reachable parts of the system and the lower levels represent parts deeper down in the system.

2. The Search Algorithm

As in [1] the troubleshooting is incremental. At every step of the troubleshooting process, based on the current information and on the system model, the best action is calculated with the aim of minimizing the expected cost of repair. The system is considered to be repaired when the probability in the belief state for the fault free state is approximately one.

In our algorithm the choice of action is calculated by searching through alternative action plans as done in [10] and [11]. This is in contrast to [6] where the idea of deciding which action to take by searching is abandoned in favor of directly choosing the action that has the best relation between the probability of having a certain fault, given the information at hand, and the cost of performing the action. In this paper, as in [2], we will call this relation the *efficiency* of an action. When choosing actions based on efficiencies the computational complexity is significantly lower compared to complete searching. Even though, a complete AND/OR search is exponential in time, with heuristics that limit the branching factor and search depth, the search can be performed in reasonable time and still provide satisfactory results. With a precise heuristic value, forward pruning can be made with only a small loss in optimality.

2.1. The AND/OR Tree Representation

For each fault revoking action it is possible to calculate a single new belief state and for each testing action it is possible to calculate the resulting belief state for each possible outcome of the test. The relation between belief states, actions, and observations can be described as an AND/OR tree [12]. It is a directed tree with nodes of two types: OR nodes representing alternative ways of solving the problem, and AND nodes representing problem decomposition into subproblems, all of which need to be solved [13].

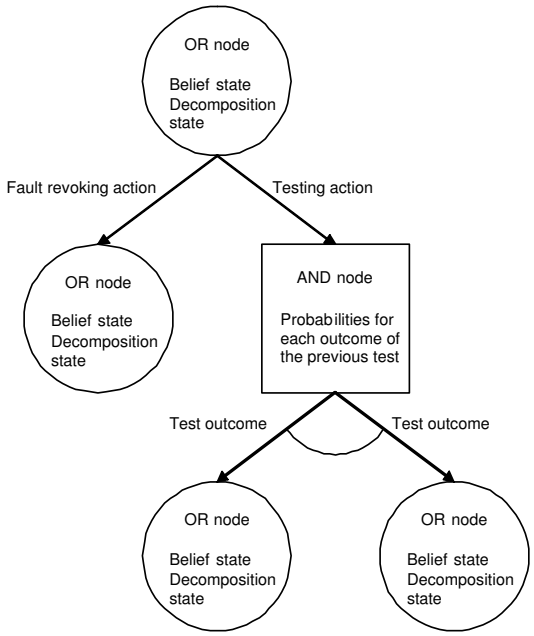

Figure 2. OR nodes have one child for each *applicable action* given the belief state of the node. AND nodes have one child for every possible outcome of the corresponding testing action. The children of an AND node are always OR nodes.

A fault revoking action defines the edge between two OR nodes and a testing action defines the edge from an OR node to an AND node. The test outcomes define the edges from an AND node to its children. Every OR node is labeled with a belief state and a decomposition state and every AND node is labeled with the probabilities for each outcome of the previous testing action. The root node of the AND/OR tree is always an OR node containing the initial belief state and initial decomposition state. Figure 2 illustrates the structure of the AND/OR tree.

2.2. Minimal Expected Cost of Repair

The search algorithm finds the choice of actions that minimizes the expected cost of repair. Let $c(n, m)$ be the cost of performing the action on the edge between the OR node n and its child m. Included in this cost are the costs of making the necessary transitions in the decomposition state. Let $p(n, m)$ be the probability of the test outcome associated with the child m of the AND node n. A goal node is an OR node where the system is in a fault free state. When a goal node is reached all that needs to be done is to restore the system to a fully assembled state. Let $r(n)$ be the cost of restoring the decomposition state of the goal node n to the fully assembled state. The minimal expected cost of repair for a node n is calculated as

$$C_{min}(n) = \begin{cases} r(n) & \text{if } n \text{ is an goal node} \\ \min\limits_{m \in ch(n)} \big(c(n,m) + C_{min}(m)\big) & \text{if } n \text{ is an OR node} \\ \sum\limits_{m \in ch(n)} p(n,m) C_{min}(m) & \text{if } n \text{ is an AND node} \end{cases} \quad (1)$$

Where $ch(n)$ is the set of all children of n. Further details on how (1) is derived can be found in [14].

The minimal expected cost of the entire tree is C_{min} of the root node. The goal for the search algorithm is to find the choice of actions that satisfies (1).

2.3. Searching the Tree

The algorithm searches the AND/OR tree using a recursive depth first search strategy. For every expanded node it maintains values for lower bound $lb(n)$, upper bound $ub(n)$, as well as a heuristic value $h(n)$.

The lower bound is calculated as the cost of repairing the system if one could make a "perfect test" that gains full information of the underlying fault state. Since there are no negative costs and since all faults must be repaired, $lb(n) \leq C_{min}(n)$. This method is calculating the lower bound is also used in [15].

The upper bound is the best solution found so far. The upper bound of a node is inherited from its parent and is updated as the algorithm backtracks. For children of AND nodes, the inherited upper bound corresponds to their share of the parent's upper bound,

$$ub(m) = \frac{ub(n) - \sum\limits_{\substack{m' \in ch(n) \\ m' \neq m}} p(n,m') lb(m')}{p(n,m)} \quad (2)$$

where n is the parent node of m [14]. Nodes where the lower bound is greater than the upper bound can be pruned.

When choosing which node to expand next, the children of an AND node are sorted in a descending order according to the value of the probability $p(n,m)$. The children of an OR node are sorted in an ascending order according to their heuristic value $h(n)$. If the child is an OR node, this value is calculated as

$$h(n) = lb(n) + c_{entropy} H(n) \quad (3)$$

where $H(n)$ is the *entropy* [16] of the belief state in node n and $c_{entropy}$ is a parameter which can be thought of as the mean cost of reducing entropy. AND nodes do not contain belief states so the heuristic value cannot be calculated in the same way. Instead, the heuristic value of an AND node is calculated by weighting the heuristic values of its children, which are OR nodes, with their probability:

$$h(n) = \sum_{m \in ch(n)} p(n,m) h(m) \quad (4)$$

We wish that the heuristic value is an estimate of the minimal expected cost of repair, i.e. $h(n) \approx C_{min}(n)$. Given a set of training data where the true minimal expected cost

of repair C_{min} is known for every node, the parameter $c_{entropy}$ is estimated using the least square method in (3) where $C_{min}(n)$ is substituted for $h(n)$.

So far the algorithm is guaranteed to find the optimal solution that satisfies (1), but for complex problems finding this can be computationally intractable. If the heuristic is accurate enough, the branching factor can be limited to allow forward pruning with little loss in optimality. With an additional limit to the search depth, the algorithm can be guaranteed to finish within reasonable time.

3. Performance of the Algorithm on the Fuel Injection System Model

A precondition for solving troubleshooting problems by searching is that the search algorithm finds its solution adequately fast. In [8] an existing fuel injection system of a truck is modeled as a Bayesian network [17]. The actions that can be performed on the fuel injection system are strongly dependent on its decomposition state. We let the algorithm find the optimal solution for troubleshooting the fuel injection system starting from randomly generated inputs. When using the heuristic based on entropy (3), the optimal solution was found in average among the first two branches searched [14]. This allows us to put a tight limit on the branching factor to make a deeper search.

In [9], Breese and Heckerman's decision theoretic troubleshooting algorithm [6] is evaluated in a series of experiments on the fuel injection system model. In these experiments, a Monte Carlo simulation of the complete repair process is made. The set up consists of a simulated fuel injection system that has a predefined fault. When a testing action is performed the resulting observation is generated randomly according to probabilities from the probability model. The total cost of repair is the cost of all actions performed until the system is verified to be free of faults. In each experiment a different fault is predefined and the average of the total cost of repair is measured.

We repeated the same series of experiments under the same conditions using the here proposed search algorithm. In average over all experiments, the cost of repair was 32% less. The greatest differences in costs are in cases where the faulty component is hard to isolate and the risk of making the wrong repair is high. In these cases, when a repair is made, several expensive transitions in the decomposition state have to be made to verify if the repair is correct. Our algorithm avoids this by repairing several faults before returning to a fully assembled decomposition state whenever this is beneficial. A more detailed analysis of these experiments can be found in [14].

4. Conclusions

When troubleshooting large mechanical systems such as trucks it is not always realistic to assume that the cost of performing actions are independent of previously performed actions. Therefore the troubleshooting model in [6] has been extended with the decomposition state that describes which parts of the system that are assembled or decomposed. We proposed a new troubleshooting algorithm that finds the best action to perform by searching an AND/OR tree using a heuristic based on entropy. In the experiments on the fuel injection system, we showed that the expected cost of repair can be reduced significantly when using the here proposed search algorithm compared to when using the method proposed in [6].

Acknowledgments

We would like to thank Anders Florén, Hans Ivendal, Göran Johansson, Mats Karlsson, Per Nyblom, Anna Pernestål, and Jan Sterner for many fruitful and interesting discussions. We would also like to thank Katja Lotz, Jonatan Mossberg, and Helena Sundberg for supplying us with data and algorithms needed for the experiments.

References

[1] J.S. Breese and D. Heckerman. Decision-theoretic troubleshooting: A framework for repair and experiment. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 124–132, San Fransisco, 1996. Morgan Kaufmann.

[2] H. Langseth and F.V. Jensen. Decision theoretic troubleshooting of coherent systems. *Reliability Engineering and Systems Safety*, 80(1):49–62, april 2002.

[3] Helge Langseth and Finn V. Jensen. Heuristics for two extensions of basic troubleshooting. In *Proceedings of the 7th Scandinavian Conference on Artificial Intelligence, SCAI'01, Frontiers in Artificial Intelligence and Applciations*, pages 80–89, 2001.

[4] D. Gillblad, A. Holst, and R. Steinert. Fault-tolerant incremental diagnosis with limited historical data. Technical report, Swedish Institute of Computer Science, Kista, 2006.

[5] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, 2003.

[6] D. Heckerman, J. Breese, and K. Rommelse. Decision-theoretic troubleshooting. In *Communications of the ACM 38*, pages 49–57, 1995.

[7] K. Lotz. Optimizing guided troubleshooting using interactive tests and bayesian networks with an application to diesel engine diagnosis. Master's thesis, Department of Mathematics, Royal Institute of Technology, 2007.

[8] J. Mossberg. Bayesian modeling of a diesel injection system for optimal troubleshooting. Master's thesis, Department of Mathematics, Royal Institute of Technology, 2007.

[9] H. Sundberg. Decision-theoretic troubleshooting using bayesian networks - guided troubleshooting of a diesel injection system. Master's thesis, School of Computer Science and Communication, Royal Institute of Technology, 2007.

[10] P. P. Faure, X. Olive, L. Travé-Massuyès, and H. Poulard. AGENDA: Automatic GENeration of DiAgnosis trees. In *Journées Doctorales d'Automatique JDA'01*, pages 203–214, 2001.

[11] 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.

[12] M. Ghalab, D. Nau, and P. Traverso. *Automated Planning*. Morgan Kaufmann, San Francisco, 2004.

[13] R. Dechter and R. Marinescu. And/or search spaces for graphical models. *Artificial Intelligence 171*, pages 73–106, 2007.

[14] H. Warnquist and P. Säby. Conditional planning for troubleshooting and repair in a partially observable environment. Master's thesis, Department of Computer and Information Science, Linköping University, 2008. Not yet published as of submission date.

[15] Marta Vomlelová and Jiří Vomlel. Troubleshooting: Np-hardness and solution mehtods. In *Proceedings of the Fifth Workshop on Uncertainty Processing, WUPES'2000*, 2000.

[16] R.M. Gray. *Entropy and Information Theory*. Springer, New York, 1990.

[17] Finn V. Jensen. *An Introduction to Bayesian Networks*. Springer, 1996.