Final thesis

Conditional Planning for Troubleshooting and Repair in a Partially Observable Environment

by

Håkan Warnquist  Petter Säby

LIU-IDA/LITH-EX-A--08/013--SE

2008-04-08
**Conditional Planning for Troubleshooting and Repair in a Partially Observable Environment**

**Title**

Conditional Planning for Troubleshooting and Repair in a Partially Observable Environment

**Författare**

Håkan Warnquist  
Petter Säby

**Sammanfattning**

Vehicles of today contain many advanced and complex systems, systems that make it hard for the mechanics working with them to keep an overview. In addition, new systems are introduced at an increasingly higher pace, which makes it hard or impossible for the mechanics to keep a both broad and deep competence. Consequently, to maintain a fast and efficient repair process, there is a need for computer-aided diagnosis.

In this thesis we develop a method for choosing the best *next action* in a repair process, using observations and a probability model. We describe the state of the system as a belief-state, a probability distribution over the faults that can occur on the system. An AND/OR-tree is used when searching for the optimal repair plan. We use entropy to speed up the algorithms. To avoid expensive validation actions, the system functionality is only inspected if the probability of having a fault free system is above a certain level.

The method is compared with two implementations of an existing method, with good results. The method can favorably be used on systems with many possible faults.
Final Thesis

Conditional Planning for Troubleshooting and Repair in a Partially Observable Environment

by

Håkan Warnquist  Petter Säby

LIU-IDA/LITH-EX-A--08/013--SE

2008-04-08

Supervisor: Per Nyblom
Examiner: Jonas Kvarnström
Abstract

Vehicles of today contain many advanced and complex systems, systems that make it hard for the mechanics working with them to keep an overview. In addition, new systems are introduced at an increasingly higher pace, which makes it hard or impossible for the mechanics to keep a both broad and deep competence. Consequently, to maintain a fast and efficient repair process, there is a need for computer-aided diagnosis.

In this thesis we develop a method for choosing the best next action in a repair process, using observations and a probability model. We describe the state of the system as a belief-state, a probability distribution over the faults that can occur on the system. An AND/OR-tree is used when searching for the optimal repair plan. We use entropy to speed up the algorithms. To avoid expensive validation actions, the system functionality is only inspected if the probability of having a fault free system is above a certain level.

The method is compared with two implementations of an existing method, with good results. The method can favorably be used on systems with many possible faults.
Acknowledgments

We would like to thank everyone, who in some way was involved in our work, making this master thesis possible. Mattias Nyberg was our supervisor at Scania. He is the person we had most contact with during our work, coming with challenging ideas and demands during our work. Per Nyblom, our supervisor at Linköping University, gave us loads of useful feedback. Our examiner Jonas Kvarnström helped us improve the report even further. The Reference Group at Scania, consisting of Anders Florén, Anna Pernestål, Hans Ivendal, Jan Sterner, Jonas Biteus and Mats Karlsson, was always bringing us positive criticism. Earlier students, writing their master theses at Scania, helping us out with the evaluation of our methods. Then there is all the other people of NED and YSKP, making our everyday life at Scania joyful.

Thank you all!
## Contents

1 Introduction ................................................................................. 3  
1.1 Background ............................................................................ 3  
1.2 Problem Formulation .............................................................. 4  
1.2.1 Assumptions ........................................................................ 4  
1.2.2 Goal ...................................................................................... 4  
1.2.3 Problem Properties and Modeling Decisions ..................... 4  
1.3 System Overview ................................................................. 5  
1.4 Thesis Outline ........................................................................ 6  

2 Probability and Bayesian Networks ........................................... 7  
2.1 A Bayesian Network .............................................................. 7  
2.1.1 Full Joint Probability .......................................................... 9  
2.1.2 Conditional Probability ...................................................... 9  
2.1.3 Conditional Independence - Independency of Children .... 9  
2.1.4 Conditional Dependence - Dependency of Parents ........... 10  
2.2 Modeling the Bicycle Example ............................................ 11  

3 System Modeling ......................................................................... 13  
3.1 Representation ........................................................................ 13  
3.1.1 Faults and Observations .................................................... 13  
3.1.2 Actions .............................................................................. 14  
3.2 Two Layer Network ................................................................ 15  
3.3 Reducing the System ............................................................ 15  
3.4 Belief-State ............................................................................ 16  
3.4.1 Updating Belief-State from Observations ....................... 17  
3.4.2 Updating Belief-State after Fault Revoking Actions ....... 17  
3.5 Decomposition State ............................................................. 18  
3.5.1 Making Transitions in the Decomposition State ............ 20  
3.5.2 Cost of Decomposition State Transitions .................... 22  

4 Full Search .................................................................................. 25  
4.1 Repair Policies ....................................................................... 25  
4.2 Expected Cost of a Repair Policy ......................................... 26  
4.3 The Policy Generator ............................................................ 27  
4.4 A Simple Example .................................................................. 29
Chapter 1

Introduction

In a future, not too long from now. In a workshop, not too far from here. A truck driver brings his vehicle in, informing the mechanic about a series of problems that he needs help with. The mechanic connects his computer to the vehicle and lets a diagnosis program download data from the vehicle computer. Then he helps the truck driver to describe the problems more precisely, and inserts his observations into the diagnosis program. Using this data as input, together with a mathematical model of the vehicle, the diagnosis program now gives suggestions on how to solve the problems as fast, easy and cost efficient as possible. This report investigates how such a program can be made, using probability theory.

1.1 Background

Vehicles of today contain many advanced and complex systems, systems that make it hard for the mechanics working with them to keep an overview. In addition, new systems are introduced at an increasingly higher pace, which makes it hard or impossible for the mechanics to keep a both broad and deep competence. Consequently, to maintain fast and efficient repair, there is a need for computer-aided diagnosis.

This thesis work was done at the Division for Service Methods Framework Development (YSKP) and the Division for Power Train Control System Diagnosis (NED) at Scania, based in Södertälje, Sweden. YSKP is responsible for service and repairs. They are interested in performing repairs as quickly, cheaply and efficiently as possible, to be able to decrease costs and to increase customer satisfaction. One way of doing this would be using automated diagnosis. NED works with on-board, law driven diagnosis, i.e. fault detection and fault isolation in the vehicle to be able to meet governmental regulations regarding emissions. This, too, can be done using automated diagnosis.

Therefore Scania is interested in obtaining more knowledge of how automated diagnoses can be done.
1.2 Problem Formulation

A truck is a complex system, and to model such a system a number of assumptions and modeling decisions have to be made. The problem can be stated under these assumptions, which strongly affect the properties of the problem.

1.2.1 Assumptions

Assume that a probability model of the system is available, or possible to estimate. The core of the model are connections between faults and observations, both being stochastic variables. The probabilities in the model are à priori probabilities for faults, and probabilities for observations conditional on the faults.

Further, assume that actions can be carried out on the system, to change the mode of the system or to gain more knowledge about the system, e.g. a repairing action that would change the fault state of the system, or a testing action, that would give us more knowledge about what faults might be at hand.

Finally, assume that a cost function for performing actions exists. The cost function could symbolize time, money, some other quantity, or a combination thereof.

1.2.2 Goal

We want to find the first action in a repair plan that minimizes the cost of fully repairing the system. To do this, a framework needs to be set up, and an algorithm for finding that repair plan needs to be found, see Figure 1.1.

Figure 1.1: Main algorithm. The input to the algorithm consists of observations, a model of the system that is to be diagnosed, and a cost function. The output consists of a repair plan. The action to perform is the first action of that repair plan.

1.2.3 Problem Properties and Modeling Decisions

Some modeling decisions that characterize the problem are

- an action that tries to revoke a fault always succeeds
• observations are modeled as random variables
• performing a test several times will not increase the certainty or accuracy of the test\(^1\)
• the aim is to reach a goal state (a fault free state)

Some properties that are specific to problem instances of troubleshooting and repairing a truck are
• the problem has a large state space, but a small support\(^2\)
• the support varies from case to case, depending on the observations
• there is a large range of actions that can be performed on the system, but in each state only a few are of interest, depending on the support
• the large amount of variables leads to a large amount of calculations, and to find a solution in reasonable time simplifications have to be made
• the cost of performing an action strongly depends on to what extent the truck has been taken apart
• the functional control of a truck is expensive, relative to the cost of performing repair actions

1.3 System Overview

At the beginning of the troubleshooting observations and sensor data are inserted to the diagnosis software, the so called troubleshooter, and an action is proposed for the mechanic to perform. After performing the action, or if the mechanic at any point during the troubleshooting process makes new observations, observations can be inserted to the troubleshooter. Updating the troubleshooter in this way may lead to different actions being suggested. See figure 1.2 for further illustration. We quit the cycle when there is sufficient belief that the system is repaired.

\(^1\)For example, measuring the voltage between two points in an electrical circuit twice is assumed to deliver the same result both times, unless there have been changes to the circuit in between the measurements. Measuring twice would not increase our belief in the test result.

\(^2\)There can be thousands of possible faults on the system, but given the current observations only a few of them are of interest. For each troubleshooting case there might be different observations, and so the faults of interest change. We use the term support for these faults of interest.
1.4 Thesis Outline

After the background of the problem, and the problem formulation, in Chapter 1, an introduction to probability theory and Bayesian networks are given in Chapter 2, providing the basic theory required. Founded on this theory, the modeling of the problem follows in Chapter 3, enabling the two different suggested solution methods of Chapter 4 and 5. Two existing methods are presented and discussed in Chapter 6, and together with the test case modeled in Chapter 7 this enables the evaluation of the solution methods in Chapter 8. Following on this, our conclusions are given in Chapter 9. Finally, ideas for future improvements are presented in Chapter 10.

Throughout the report an example will evolve, applying the theory on a simple model of a bicycle. Whenever referred to the example, this is the referred example.

Figure 1.2: System overview.
Chapter 2

Probability and Bayesian Networks

Using probability theory in troubleshooting requires a model of the system that is to be diagnosed. In this work it is assumed that a model is given in the form of a Bayesian network. We do not discuss whether or not it is possible to make such a model of a complex system like a truck\(^1\). This chapter introduces Bayesian networks in the extent needed to understand the mathematics in later chapters.

2.1 A Bayesian Network

A Bayesian network is a way of graphically representing how variables are connected with, and influence, each other. It consists of two parts: a directed, acyclic graph, and conditional probability distributions. The graph consists of nodes representing variables, and edges illustrating how the variables are connected. Or rather, missing edges represent irrelevant connections [2, p. 27]. The conditional probability distributions are here given in conditional probability tables (CPTs).

If two nodes are connected with an arrow, the node at the beginning of the arrow is denoted parent node, and the node at the end of the arrow is the child node. A parent node can have several child nodes, and a child node can have several parent nodes.

A CPT gives the probability distribution for the states of a variable \(X\), conditional on the states of its parents. The parents of a node \(X\) are denoted \(pa(X)\), and for a node representing a variable with the two possible states \(x_1\) and \(x_2\), and with two binary parents, the CPT is read as following:

\(^1\)However, a Bayesian network model of an injection system has been made in an earlier master thesis at Scania (the XPI-system is modeled in [14]), and the matter of how to obtain probabilities that can be used in Bayesian network models is subject of ongoing research at Scania [15].
In other words, in a Bayesian network it is assumed that the probability for a variable is only conditional on its parents,

\[ P(X|\text{all other variables}) = P(X|\text{pa}(X)) \]   \hspace{1cm} (2.1)

Since for a variable with the two states \( x_1 \) and \( x_2 \) we get \( P(X = x_2|\text{pa}(X)) = 1 - P(X = x_1|\text{pa}(X)) \), it is enough to show one of the two values in the CPT. This will be the case in some of the examples. For a variable that has no parents, the CPT contains the à priori probabilities for the variable. We illustrate using an example, inspired by an example in [11].

**Example**

The wet grass on a lawn can be caused by rain or by the sprinkler system. If we know that it is raining, the probability of having wet grass is very high. Oppositely, the knowledge of having wet grass would increase the probability of rain, as well as the probability that the sprinklers are on. See Figure 2.1.

<table>
<thead>
<tr>
<th>States of the parents</th>
<th>( X = x_1 )</th>
<th>( X = x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>( P(X = x_1</td>
<td>\text{pa}(X) = 00) )</td>
</tr>
<tr>
<td>0 1</td>
<td>( P(X = x_1</td>
<td>\text{pa}(X) = 01) )</td>
</tr>
<tr>
<td>1 0</td>
<td>( P(X = x_1</td>
<td>\text{pa}(X) = 10) )</td>
</tr>
<tr>
<td>1 1</td>
<td>( P(X = x_1</td>
<td>\text{pa}(X) = 11) )</td>
</tr>
</tbody>
</table>

Figure 2.1: Wet grass example
2.1.1 Full Joint Probability

For a network with \( n \) variables \( X_i \), the full joint probability distribution \( P(X_1, \ldots, X_n) \) is

\[
P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\text{pa}(X_i)), \quad (2.2)
\]

that is, the probability distribution for all variables in the network can be obtained as the product of the probabilities for the individual variables, conditional on their parents. The equation is fundamental in Bayesian networks, and from it we can obtain the probability for any state of the system. This applies for conditional as well as for unconditional probabilities. All that is needed is the CPTs, the conditional probability distributions for the individual variables.

2.1.2 Conditional Probability

Why are Bayesian networks and conditional probabilities of interest? We try to model a system, on which faults can occur. Unfortunately, we do not have full knowledge of these faults. The faults can lead to certain observations, and of the observations we do have knowledge. Further, we assume that we know, or can estimate, the probability for an observation, conditional on one or several faults,

\[
P(\text{observation}|\text{faults}) \quad (2.3)
\]

From the knowledge of the observations, and the knowledge of how faults and observations are connected, we can now increase our knowledge, or belief, about the faults. This is done using Bayes’ theorem. If \( A \) and \( B \) are two events, Bayes’ theorem can be posed as

\[
P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.4)
\]
or in the terms we are using

\[
P(\text{faults}|\text{observations}) = \frac{P(\text{observations}|\text{faults})P(\text{faults})}{P(\text{observations})} \quad (2.5)
\]

In this way we can increase our knowledge about the faults, using knowledge about observations.

2.1.3 Conditional Independence - Independency of Children

In the Bayesian network we are interested in, the variables are random variables. Each variable can, with a certain probability, take on a certain state. It is of importance for the calculations in this report when variables are independent, and when they are not.

If a node has several child nodes, knowledge about the state of one of the child nodes influences the probability for the other nodes. However, this is only valid as long as the state of the parent node is unknown. If the state of the parent node is known, the child nodes are independent. This is favorably illustrated with an example, this one originally found in [14].
Example

A person can be Male or Female, and has the attributes Stature and Hair length. The stature can be either Tall or Short, and the Hair length either Long or Short. As long as the gender of the person is unknown, the knowledge that the person is short would increase the probability that the person has long hair. This shows a dependency between the child nodes, and is due to the fact that male persons on average are taller than females, and that females more often than male persons have long hair. If on the other hand the gender of a person is known to be female, the knowledge about the person’s length does not influence the belief that the person has long hair. The belief that the person has long hair only directly depends on the gender, that now is known to be female.

Figure 2.2: Independency of Children

Put in mathematical terms, using G for gender, S for stature, and H for hair length, the conditional independence is written

\[ P(S|G,H) = P(S|G) \quad (2.6) \]

if S and H are independent, conditional on G. This leads to

\[ P(S,H|G) = P(S|G)P(H|G), \quad (2.7) \]

whereas in the general case

\[ P(S,H) \neq P(S)P(H) \quad (2.8) \]

The conditional independence property is important, and will be used when modeling the system in Chapter 3.

2.1.4 Conditional Dependence - Dependency of Parents

Two parent nodes with a common child node, but with no direct connections, are independent. The state of one of the parents does not affect the probability for the other parent node. However, this is only valid as long as the state of the child node is unknown. If we gain knowledge of the state of the child node, the parents are suddenly dependent. Gaining knowledge of the state of one of the parents would now affect the probability for the other parent node.
Example

We return to the example with the wet grass. The wet grass can be caused either by rain or by the sprinkler system. It is also possible that the sprinkler system is on during rain. As long as nothing is known about the state of the grass, the probabilities for rain and the sprinkler system are independent. If we know that the grass is wet, the probability for rain as well as for the sprinkler system being on increases. But if we would now find out that the sprinkler system is on, the belief in rain would decrease again.

![Figure 2.3: Parental dependencies](image)

Put in equations, now using $R$ for rain, $S$ for sprinkler, and $W$ for wet grass, the conditional dependence can be written

$$P(R|S,W) \neq P(R|W) \quad (2.9)$$

or, in a way more similar to the equations in the model,

$$P(R,S|W) \neq P(R|W)P(S|W), \quad (2.10)$$

denoting that $R$ and $S$ are dependent, conditional on $W$, whereas we still have the original independencies

$$P(R|S) = P(R) \quad (2.11)$$
$$P(R,S) = P(R)P(S) \quad (2.12)$$

Also the conditional dependence property will be used for the system modeling in Chapter 3.

2.2 Modeling the Bicycle Example

To get more acquainted with the Bayesian networks, we model the wheel of a bicycle. The system consists of the two faults *hole on inner tube* and *broken valve* and the two observations *visible hole on inner tube* and *flat tire*. The probability for the observation *visible hole on inner tube* is conditional only on *hole on inner tube*, whereas *flat tire* is conditional on both faults. We obtain the Bayesian network in Figure 2.4.
Figure 2.4: Modeling a bicycle wheel as a Bayesian network.
Chapter 3

System Modeling

In this chapter we present the representation used in the model, and a description of the mathematics for this problem is given.

3.1 Representation

3.1.1 Faults and Observations

For a system there is a limited set of faults that can occur. A fault is something that is broken on the system and that needs to be repaired, replaced or adjusted, e.g., a broken inner tube or valve on the wheel of a bicycle.

Definition 1 (System Faults). The system faults, \( F \), are an ordered finite set of variables, where each variable represents a fault that may occur on a system. Each \( F_i \in F \) has the domain \( \Omega_{F_i} = \{ \text{inactive, active} \} \). The set \( F \) has the domain \( \Omega_F = \Omega_{F_1} \times \cdots \times \Omega_{F_k} = \{(f_1, \cdots, f_k) | f_i \in \Omega_{F_i}, F_i \in F, k = |F| \} \).

Example

\[
F = (F_{\text{hole}}, F_{\text{valve}})
\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_{\text{hole}} )</td>
<td>Hole on inner tube</td>
</tr>
<tr>
<td>( F_{\text{valve}} )</td>
<td>Broken valve</td>
</tr>
</tbody>
</table>

Given the state of the faults, the probabilities of making certain observations are known. An example could be that if the valve is broken, the tire is always flat. If there is a hole on the inner tube, the tire is always flat, and in 60% of the cases the hole is visible.

Definition 2 (Observables). The observables, \( O \), is a finite set of variables, where each variable represents an observable quantity on a system. Each \( O_i \in O \) has the domain \( \Omega_{O_i} = \{ o_1, \ldots, o_{n_i} \} \), where \( n_i \) is the number of possible outcomes of \( O_i \).
Example

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_{\text{tire}}$</td>
<td>Flat tire</td>
<td>has pressure, flat</td>
</tr>
<tr>
<td>$O_{\text{tube}}$</td>
<td>Visible hole on inner tube</td>
<td>no visible hole, visible hole</td>
</tr>
</tbody>
</table>

Note that not all observables are observed at all times.

**Definition 3** (Observations). A set of observations, $\Theta = (O_1, \ldots, O_n) \subseteq \mathcal{O}$, is an ordered set of observables that are observed. It has the domain

$$\Omega_\Theta = \Omega_{O_1} \times \cdots \times \Omega_{O_n} = \{(o_1, \ldots, o_n) | o_1 \in \Omega_{O_1}, \ldots, o_n \in \Omega_{O_n}\}.$$ 

**Example**

If the two observables $O_{\text{tire}}$ and $O_{\text{tube}}$ are observed we get

$$\Theta = (O_{\text{tire}}, O_{\text{tube}})$$

$$\Omega_\Theta = \{(\text{has pressure, no visible hole}), \quad (\text{has pressure, visible hole}), \quad (\text{flat, no visible hole}), \quad (\text{flat, visible hole})\}$$

### 3.1.2 Actions

On a system, actions can be performed. This could be changing the inner tube, measuring the pressure in the tire, etc. We divide actions into two different categories: Fault revoking actions and testing actions.

**Definition 4** (Revoke). An action $a$ **revokes** a fault $F \in \mathcal{F}$ if

$$P(F = \text{inactive}|a \text{ has been performed}) = 1$$

For the conditional probability of an event $e$, given that an action $a$ has been performed, the shorter notation $P(e|a)$ will be used instead of $P(e|a \text{ has been performed}).$

**Definition 5** (Fault Revoking Action). An action $a$ is a **fault revoking action** if it revokes one or more faults $F \in \mathcal{F}$. The set $\mathcal{F}_a \subseteq \mathcal{F}$ is the set of faults that the fault revoking action $a$ revokes.

**Definition 6** (Testing Action). A **testing action** $a$ is an action that generates a set of observations $\Theta_a$ when performed.

Replacing a component is an example of a fault revoking action. Examples of testing actions are measuring the air pressure or doing a function test of a component. Testing actions can not revoke faults, nor can any action cause faults.
3.2 Two Layer Network

With use of a Bayesian network the relationship between faults and observations can be modeled. We model a system as a two layer network, where the nodes in the first layer are elements in $\mathcal{F}$, and the nodes in the second layer are elements in $\mathcal{O}$. The arrows are directed from the faults to the observations, i.e. the probabilities for the observations are conditional on the faults, whereas the probabilities for the faults are à priori probabilities.

![Bayesian network representation of a general system](image)

Figure 3.1: Bayesian network representation of a general system. The faults $f_i$ constitute one layer, and the observations $o_i$ another layer.

Note that each fault is not necessarily connected to every observation. Normally, a fault is connected only to a few observations, and an observation is connected only to a few faults.

3.3 Reducing the System

When a vehicle comes in for service, it is checked for different common faults according to some check list. However, when it comes in because of a certain problem, the focus of the mechanic is to solve that problem, not to make a full service of the vehicle. As long as there are no complaints about a certain part of the vehicle, it is assumed to be working properly. For example, if a truck is in for repair because the driver has noticed a strange noise from the engine, there is no reason to believe that the head light is not working properly.

Similarly, to reduce the complexity of the problem, we assume that all parts of the system are working properly, unless there has been an observation suggesting that a certain part could be faulty.

With this assumption, we only have to consider those faults that could have caused the observations that have been made. If at a certain stage of the troubleshooting the observations $\Theta$ have been made, let the new system faults $\mathcal{F}$ be the ordered set of all faults that have direct connections to observations in $\Theta$. In Bayesian network representation, $\mathcal{F}$ are all parents of the observations,

$$\mathcal{F} = pa(\Theta)$$  \hspace{1cm} (3.1)
As a consequence of this, the observables can also be reduced. We keep only those observables that can be caused by the faults in the reduced $F$, i.e. the children of $F$.

$$O = ch(F) \quad (3.2)$$

This new, reduced, model is the model that is used for troubleshooting. Note that if an observation is made at a later stage in the repair process, that is not part of the reduced model, this suggests that other faults might be at hand. In this case, all faults connected to this observation can be re-included in the model, as well as the observations that are children of these faults. That is, no faults are permanently discarded only because they were not suggested by the initial observations.

### 3.4 Belief-State

The states of the faults of a system are not known. We introduce the notion of combination of faults to describe each possible state of the system, excluding the observations. Each combination of faults has a probability to occur. The distribution of these probabilities is called the belief-state of the system, and is obtained using the full joint distribution in Equation 2.2.

**Definition 7** (Combination of Faults). Given the system faults $F$, a variable $\varphi$ is a combination of faults if $\varphi \in \Omega_F$. Let $\varphi(i)$ denote the $i$th element of $\varphi$.

The state where the system has no active faults is denoted $\varphi_0$.

Let $I$ denote the state of information, and let it represent all knowledge acquired. It contains knowledge of what observations have been made, what actions have been performed, and in what order this has happened.

**Definition 8** (Belief-State). A belief-state is a probability distribution for all combinations of faults $\varphi \in \Omega_F$. For a system with the belief-state $b$, let $b(\varphi)$ denote the probability to have a certain combination of faults $\varphi$, conditional on the state of information $I$, $b(\varphi) = P(\varphi|I)$.

**Example**

Let us go back to the example in Figure 2.4 on page 12. The system faults are $F = (F_{hole}, F_{valve})$. Let the state of information be $I = ”No observations are made”. Since no observations have been made (or in other words, since the states of the common children are not known), the probabilities of the faults are independent. Consequently, the probability for the combination can be obtained as the product of the individual probabilities. We get the following belief-state:

<table>
<thead>
<tr>
<th>Combination of faults in $F$</th>
<th>$b(\varphi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi_0$</td>
<td>0.5 \cdot 0.8 = 0.4</td>
</tr>
<tr>
<td>(inactive, active)</td>
<td>0.5 \cdot 0.8 = 0.4</td>
</tr>
<tr>
<td>(active, inactive)</td>
<td>0.5 \cdot 0.2 = 0.1</td>
</tr>
<tr>
<td>(active, active)</td>
<td>0.5 \cdot 0.2 = 0.1</td>
</tr>
</tbody>
</table>

It is possible to derive the belief-state $b$ from $I$ at any stage of the troubleshooting, using the prior probabilities.
3.4.1 Updating Belief-State from Observations

When new information is added in the form of the observations $\theta$, a new belief-state $b'$ can be calculated as

$$b'(\varphi) = P(\varphi|I, \theta) = \frac{P(\theta|I, \varphi)P(\varphi|I)}{P(\theta|I)} \quad (3.3)$$

using Bayes’ theorem from Equation 2.4.

When modeling the system as a Bayesian network, it follows that the probabilities for observations are conditional only on their parents, i.e. the faults, see Equation 2.1. If a certain combination of faults $\varphi$ is given, the probability for the observations $\theta$ conditional on both $I$ and $\varphi$ can be reduced to

$$P(\theta|I, \varphi) = P(\theta|\varphi) \quad (3.4)$$

since $I$ merely can provide a probability distribution over $\varphi$. Further, the denominator $P(\theta|I)$ can be marginalized over the combinations of faults $\varphi$,

$$P(\theta|I) = \sum_{\varphi \in \Omega_F} P(\theta|\varphi, I) = \sum_{\varphi \in \Omega_F} P(\theta|\varphi)P(\varphi|I) \quad (3.5)$$

With Definition 8 and (3.4) and (3.5), (3.3) becomes

$$b'(\varphi) = \frac{P(\theta|\varphi)b(\varphi)}{\sum_{\varphi' \in \Omega_F} P(\theta|\varphi')b(\varphi')} \quad (3.6)$$

As a consequence of the conditional independence of Section 2.1.3, the observations $o \in \theta$ are independent, given $\varphi$. Therefore it is possible to simplify (3.4) further,

$$P(\theta|\varphi) = \prod_{o \in \theta} P(o|\varphi) \quad (3.7)$$

With (3.6) and (3.7), (3.3) becomes

$$b'(\varphi) = \frac{\left(\prod_{o \in \theta} P(o|\varphi)\right)b(\varphi)}{\sum_{\varphi' \in \Omega_F} \left(\prod_{o \in \theta} P(o|\varphi')\right)b(\varphi')} \quad (3.8)$$

3.4.2 Updating Belief-State after Fault Revoking Actions

When a fault revoking action $a$ is performed the belief-state changes. The probability mass is moved from each combination of faults $\varphi$ which have active faults that are revoked by $a$, to the combination of faults the system would have after performing $a$. Let $\varphi$ be the combination of faults before an action is performed, and $\varphi'$ the combination of faults after an action is performed. Let $T(\varphi, a)$ denote the combination of faults after a fault revoking action $a$ is performed.

**Definition 9 (The Transition Function T).** Let $A_R$ be the set of fault revoking actions, and $\mathcal{F}_a \subseteq \mathcal{F}$ be the faults that a fault revoking action $a$ revokes. The
transition function $T : \Omega \times A \mapsto \Omega$ maps each pair of combination of faults and fault revoking action $(\varphi, a)$ to a new combination of faults $\varphi'$ in such a way that for all $i$

$$\varphi'(i) = \begin{cases} 
\text{inactive} & \text{if } F_i \in F_a \\
\varphi(i) & \text{otherwise}
\end{cases}$$

This is denoted $\varphi' = T(\varphi, a)$.

If $I$ is the state of knowledge before the action is performed, and $I'$ is the new state of knowledge, the new belief-state $b'$ is retrieved through

$$b'(\varphi) = P(\varphi'|I') = P(\varphi'|a, I) = \sum_{\varphi \in \Omega_f} P(\varphi'|a, I, \varphi) P(\varphi|a, I) \tag{3.9}$$

by marginalizing over the combination of faults $\varphi$ before the action was performed. Here, $P(\varphi'|a, I)$ is the probability of having $\varphi'$ after performing $a$ when the state of knowledge was $I$. Further, $P(\varphi'|a, I, \varphi)$ can be simplified as

$$P(\varphi'|a, I, \varphi) = P(\varphi'|a, \varphi), \tag{3.10}$$

with the same argument as for Equation 3.4. The value of this probability is

$$P(\varphi'|a, \varphi) = \begin{cases} 
1 & \text{if } \varphi' = T(\varphi, a) \\
0 & \text{otherwise}
\end{cases} \tag{3.11}$$

The factor $P(\varphi|a, I)$ is the probability of having $\varphi$ when the state of knowledge is $I$ and action $a$ will be performed. Since, of course, future actions do not influence the current combination of faults, this can be simplified as

$$P(\varphi|a, I) = P(\varphi|I) = b(\varphi) \tag{3.12}$$

With (3.10), (3.11) and (3.12), (3.9) becomes

$$b'(\varphi) = \sum_{\varphi \in \Omega_f} P(\varphi'|a, \varphi) b(\varphi) = \sum_{\varphi \in \Omega_f} b(\varphi) \tag{3.13}$$

3.5 Decomposition State

Before the inner tube on the wheel of a bicycle can be exchanged, the wheel has to be dismounted from the bike and the tire has to be taken off. The time required for dismounting the wheel could be considered affecting the cost of exchanging the inner tube. Therefore, knowing whether a wheel is dismounted or not is of importance when choosing the next action in a repair process. That a wheel is dismounted does not need to be represented as a fault, since it is a fully normal part of the repair process. In addition, since the mechanic always knows if the wheel is dismounted, this does not need to be represented in a belief-state. Therefore, a decomposition state is introduced, being fully observable and representing parts of the system that could be taken apart, or in some other way be changed, during a normal repair process. Each action can require certain parts to be taken apart or to be put together before it can be performed.

An individual state, a decomposable, could be representing whether a wheel is dismounted or not, or whether an outer tire is mounted or has been taken
off, which would be required before the action \emph{Change inner tube} can be carried out. The decomposition state here is an extension of the \emph{Assembly Level} or \emph{Level of Disassembly} described in Lotz \cite{Lotz13}, Mossberg \cite{Mossberg14} and Sundberg \cite{Sundberg18}. In \cite{Lotz13}, \cite{Mossberg14} and \cite{Sundberg18} exactly one component is allowed to be decomposed at a time. Here their decomposition state is extended, allowing several parts of the system being taken apart at the same time, as long as they do not contradict each other.

\textbf{Definition 10} (Decomposition State). The decomposition state $\delta$ is a set of variables, where each variable $d \in \delta$ represents a decomposable unit of the system. Each variable is said to be in one of the modes \emph{decomposed} or \emph{assembled}.

Each action may require one or more of the decomposables to be in a certain mode. Further, each decomposable of the decomposition state may require other decomposables to be decomposed or assembled for it to be in a certain mode. For example, the inner tube on a bicycle can only be dismounted if the outer tire is already dismounted, and analogously, the outer tire can only be mounted on the wheel if the inner tube is already mounted. Mounting the wheel on the bike would only explicitly require the outer tire to be mounted, since this implies that the inner tube also is mounted.

An easy way to represent this is using a directed acyclic graph, see Figures 3.2 and 3.3. Each node represents a 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. Each node requires its parents to be in the mode \emph{decomposed}, before it can be decomposed itself. Before a node can be assembled, all its children have to be in the mode \emph{assembled}.
3.5.1 Making Transitions in the Decomposition State

Let \( \delta \) represent a certain decomposition state, e.g., \( \delta = (d_1 = \text{decomposed}, d_2 = \text{assembled}, \ldots) \). Making transitions in the decomposition state can favorably be illustrated as recursive algorithms. Before a decomposable is assembled, its children must be ensured to be in the mode \( \text{assembled} \). Similarly, before one
can be decomposed, its parents must be ensured to be in the mode *decomposed*.

If a decomposable $d$ is required to be in the mode *assembled*, and the decomposition state is $\delta$, this can be done using Algorithm 1. Analogue for decomposition, see Algorithm 2. Before an action is performed, decomposables can be set to be *assembled* or *decomposed* according to the requirements of that specific action.

**Algorithm 1: Assemble**

Name: assemble  
Input: $\delta, d$  
Output: new $\delta$

if $d$ is decomposed in $\delta$ then
  for all children $d'$ of $d$ do
    $\delta = \text{assemble}(\delta, d')$
  end
  set $d$ to assembled in $\delta$
else
  do nothing, since $d$ is already assembled
end
return $\delta$

**Algorithm 2: Decompose**

Name: decompose  
Input: $\delta, d$  
Output: new $\delta$

if $d$ is assembled in $\delta$ then
  for all parents $d'$ of $d$ do
    $\delta = \text{decompose}(\delta, d')$
  end
  set $d$ to decomposed in $\delta$
else
  do nothing, since $d$ is already decomposed
end
return $\delta$
3.5.2 Cost of Decomposition State Transitions

If a cost is introduced for making a transition in the decomposition state, the cost \( C_{tot}(dcmp \ d) \) of bringing a decomposable unit \( d \in D \) to the mode \textit{decomposed} depends on whether its parents are decomposed or assembled, and whether the parents’ parents are decomposed, etc. Consequently, to calculate the cost of bringing a certain decomposable \( d \) to the mode \textit{decomposed}, we need to keep track of all those decomposables that need to be brought to the mode \textit{decomposed} to allow this transition. We can do this by gathering all those decomposables in the set \( M(d) \) in the following way.

\[
M(d) = \begin{cases} 
\emptyset & \text{if } d \text{ decomposed} \\
\{d\} \cup \bigcup_{d' \in \text{pa}(d)} M(d') & \text{otherwise} 
\end{cases} \quad (3.14)
\]

Now let \( C(dcmp \ d) \) be the cost of bringing \( d \) from assembled to decomposed if all its parents are already in the mode \textit{decomposed}. The total cost \( C_{tot}(dcmp \ d) \) is calculated as

\[
C_{tot}(dcmp \ d) = \sum_{d' \in M(d)} C(dcmp \ d') \quad (3.15)
\]

The costs for for assembling, \( C(asmb \ d) \) and \( C_{tot}(asmb \ d) \), are obtained analogously. Calculating these costs within the Algorithms 1 and 2 gives the Algorithms 3 and 4.

**Name**: assemble with cost

**Input**: \( \delta, d \)

**Output**: new \( \delta \), cost of transition

\[
cost = 0 
\]

\[
\text{if } d \text{ is decomposed in } \delta \text{ then} \\
\quad \text{for all children } d' \text{ of } d \text{ do} \\
\quad \quad \delta, \ cost' = \text{assemble} (\delta, d') \\
\quad \text{add cost' to cost} \\
\text{end} \\
\quad \text{set } d \text{ to assembled in } \delta \\
\quad \text{add } C(asmb \ d) \text{ to cost} \\
\text{else} \\
\quad \text{do nothing} \\
\text{end} \\
\text{return } \delta, \ cost 
\]

**Algorithm 3**: Assemble with costs
Name: decompose with cost  
Input: $\delta, d$  
Output: new $\delta$, cost of transition  

cost = 0  
if $d$ is assembled in $\delta$ then  
  for all parents $d'$ of $d$ do  
    $\delta, cost' = \text{decompose}(\delta, d')$  
    add $cost'$ to $cost$  
  end  
set $d$ to decomposed in $\delta$  
add $C(dcmp\, d)$ to $cost$  
else  
  do nothing  
end  
return $\delta, cost$  

Algorithm 4: Decompose with costs
Chapter 4

Full Search

We are trying to find a repair plan that minimizes the cost of fully repairing the system. The first method developed for finding that plan is quite easy and straightforward. All possible, “reasonable” combinations of actions are created, and the expected cost for each and every one is calculated. The optimal repair plan is the combination with the lowest expected cost.

4.1 Repair Policies

A system is repaired through a sequence of actions. Whenever a testing action is performed, new information is added, and the following sequence may depend on the outcome of the test. To allow the sequence to adapt to new information we describe the repair policy as a decision tree. A decision tree is a directed tree with actions on the nodes. Nodes consisting of a fault revoking action have exactly one child since the behavior of the system for those actions are predictable. Nodes consisting of a testing action have as many children as there are possible outcomes of the test. We also introduce a terminal node indicating when a policy should terminate. This node has no children.

Definition 11 (Repair Policy). A directed tree \((V,E)\) is a repair policy if each node \(v \in V\) is either an action or a terminal node and

\[
\forall v \in V : |ch(v)| = \begin{cases} 
|\Theta_a| & \text{if } v \text{ is the testing action } a \\
1 & \text{if } v \text{ is a fault revoking action } \\
0 & \text{if } v \text{ is a terminal node}
\end{cases}
\]

where \(ch(v)\) is the set containing the children of node \(v\).

Example:

For a system there exist three possible actions; two fault revoking actions \(r_1\) and \(r_2\), and the testing action \(t\) with two possible outcomes. A terminal node is indicated as \(*\).

<table>
<thead>
<tr>
<th>a)</th>
<th>b)</th>
<th>c)</th>
<th>d)</th>
<th>e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r_1)</td>
<td>(t)</td>
<td>(r_1)</td>
<td>(r_2)</td>
<td>(r_1)</td>
</tr>
<tr>
<td>(t)</td>
<td>(r_1)</td>
<td>(t)</td>
<td>(r_2)</td>
<td>(r_2)</td>
</tr>
<tr>
<td>(*)</td>
<td>(*)</td>
<td>(*)</td>
<td>(*)</td>
<td>(*)</td>
</tr>
</tbody>
</table>
The trees a and d are repair policies according to Definition 11 but the trees b, c and e are not.

**Definition 12 (Successful Policy).** A repair policy that when executed for any fault combination \( \varphi \) of a given set of faults \( F \) puts the system in a fault-free state whenever a terminal node is reached is called *successful* for \( F \).

### 4.2 Expected Cost of a Repair Policy

A system is repaired through a sequence of fault revoking actions and testing actions, i.e. by following one path of a repair policy. However, which actions will be performed, i.e. which path will be taken, is not known in advance. This is the case since the outcomes of the tests decide which path will be taken. This means that the cost of a repair policy can not be calculated in advance. Using the probability for a certain path to be taken, the *expected cost* can be calculated.

The actual cost of executing a repair policy \( \pi \) is the sum of the costs of all actions in the path from the root to a terminal node. Since the outcome of the tests are random, the cost can also be treated as a random variable, \( C(\pi) \). The expected cost is the sum of the costs of all paths \( \rho \) in the policy, weighted with the probability that each path is taken.

\[
E\{C(\pi)\} = \sum_{\forall \rho \in \pi} C(\rho)P(\rho),
\]

with \( C(\rho) \) being the sum of the costs of all actions, and required decomposition state transitions, in the path. The probability \( P(\rho) \) depends on the probabilities of the outcomes of all tests in the path, which in return depend on the combination of faults \( \varphi_t \) when each test \( t \) is performed. If \( \varphi \) is the initial combination of faults, and \( \varphi \) is assumed to be known, the combination of faults will be known throughout the entire path, including \( \varphi_t \). This is due to the transition function \( T \). Assuming that the initial combination of faults is known, we now have

\[
E\{C(\pi)|\varphi\} = \sum_{\forall \rho \in \pi} C(\rho)P(\rho|\varphi),
\]

with

\[
P(\rho|\varphi) = \prod_{t \in \rho} P(\theta_t|\varphi_t),
\]

where \( \theta_t \) is the outcome of each test \( t \) in the path. The required probabilities \( P(\theta_t|\varphi_t) \) are obtained from the Bayesian network.

The expected cost of a policy \( \pi \) for a system with the system faults \( F \) and a certain belief-state \( b \) is

\[
E\{C(\pi)|b\} = \sum_{\varphi \in \Omega_f} E\{C(\pi)|\varphi\} b(\varphi)
\]

**Definition 13 (Optimal Policy).** A repair policy is *optimal* for a given belief-state \( b \) and a given set of faults \( F \) if it has the lowest expected cost of all policies that are successful for \( F \).
4.3 The Policy Generator

The repair policy $\pi^*$ with the smallest expected cost can be retrieved by calculating the expected cost of all repair policies in a set of relevant repair policies. For a given set of faults $\mathcal{F}$, with the knowledge that the observations $\Theta$ have been made, we wish to find a set of successful repair policies, that is small, but still includes $\pi^*$. To obtain a set of repair policies an algorithm has been developed, called the policy generator. The algorithm builds trees by adding actions from the sets $A_F$, the set of all fault revoking actions, and $A_T$, the set of all testing actions. It is called with two parameters, a set $F_r$, representing the faults that shall be guaranteed to be revoked, and a set $O_c$, representing the observations that currently have been made. It returns a set of repair policies, $\Pi$. The algorithm is recursive, and at the first calling of the algorithm $F_r = \mathcal{F}$ and $O_c = \Theta$. Pseudo code for the algorithm is given in Algorithm 5.
**Name**: Policy Generator  
**Input**: A set $F_r$ of faults, a set $O_c$ of observations, and the Bayesian network for the system.  
**Output**: A set $\Pi$ of repair policies  

If $F_r$ is an empty set then  
Return a terminal node;  
else  
Let $\Pi$ be an empty set;  
Let $A_t$ be the set of testing actions that observe child nodes of $F_r$ that are not already in $O_c$, i.e. the testing actions that can give us new information about the faults in $F_r$;  
Let $A_f$ be the set of fault revoking actions that can revoke faults in $F_r$;  
for all actions $a$ in $A_t$ do  
for all possible test results $\theta$ do  
Let $O'_c$ be the new current observations, i.e. $\theta$ added to the current observations $O_c$;  
Let $F'_r$ be the new relevant faults, by excluding all faults in $F_r$ that are impossible according to the observations $\theta$;  
Make a recursive call to Policy Generator with the input $F'_r, O'_c$, and put the resulting trees in a variable $\Pi_\theta$;  
end  
Create all combinations of trees with one tree from each $\Pi_\theta$, and let each such combination form the branches in a tree beginning with $a$;  
Add these trees to $\Pi$;  
end  
for all actions $a$ in $A_f$ do  
Let $F'_r$ be the new relevant faults, by excluding the faults $F_a$ that are revoked by $a$ from $F_r$;  
Let $O'_c$ be the new current observations, by excluding the children of $F_a$ from the current observations $O_c$;  
Make a recursive call to Policy Generator with the input $F'_r, O'_c$, and put the resulting trees in a variable $\Pi_a$;  
Add $a$ as the root of all policies in $\Pi_a$;  
Add all policies in $\Pi_a$ to $\Pi$;  
end  
Return $\Pi$;  
end  

**Algorithm 5**: Policy Generator: The algorithm creates all "reasonable" repair policies that in different ways repair the faults $F_r$.  

---  

28
4.4 A Simple Example

The theory is illustrated by a small bicycle example. The specifications for the faults, observables and actions are shown in Table 4.1. Figure 4.1 shows the graph of the Bayesian network, and Table 4.2 shows the CPTs for the observations.

| Faults    | \( P(F = \text{active}|I) \) |
|-----------|-------------------------------|
| \( F_{\text{hole}} \) | Hole on inner tube 0.2 |
| \( F_{\text{valve}} \) | Broken valve 0.5 |
| \( F_{\text{light}} \) | Broken head light 0.4 |
| \( F_{\text{dynamo}} \) | Broken dynamo 0.1 |

<table>
<thead>
<tr>
<th>Observables</th>
<th>Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O_{\text{tire}} )</td>
<td>Flat tire has pressure, flat</td>
</tr>
<tr>
<td>( O_{\text{tube}} )</td>
<td>Visible hole on inner tube no visible hole, visible hole</td>
</tr>
<tr>
<td>( O_{\text{light}} )</td>
<td>Weak or no light showing light showing, no light showing</td>
</tr>
<tr>
<td>( O_{\text{voltage}} )</td>
<td>Low voltage from dynamo voltage OK, voltage low</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fault revoking actions</th>
<th>( \mathcal{F}_a )</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{\text{tube}} )</td>
<td>Repair inner tube ( F_{\text{hole}} )</td>
<td>100</td>
</tr>
<tr>
<td>( r_{\text{valve}} )</td>
<td>Change valve ( F_{\text{valve}} )</td>
<td>30</td>
</tr>
<tr>
<td>( r_{\text{light}} )</td>
<td>Change light bulb ( F_{\text{light}} )</td>
<td>40</td>
</tr>
<tr>
<td>( r_{\text{dynamo}} )</td>
<td>Change dynamo ( F_{\text{dynamo}} )</td>
<td>75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing actions</th>
<th>( \Theta_a )</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{tire}} )</td>
<td>Check air pressure in tire ( O_{\text{tire}} )</td>
<td>40</td>
</tr>
<tr>
<td>( t_{\text{tube}} )</td>
<td>Examine inner tube ( O_{\text{tube}} )</td>
<td>20</td>
</tr>
<tr>
<td>( t_{\text{light}} )</td>
<td>Check light ( O_{\text{light}} )</td>
<td>10</td>
</tr>
<tr>
<td>( t_{\text{voltage}} )</td>
<td>Measure voltage from dynamo ( O_{\text{voltage}} )</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 4.1: Specifications for the bicycle example.
When the bicycle enters the workshop the observation $O_{\text{tire}} = \text{flat}$ is made immediately. The set of observations is $\Theta = (O_{\text{tire}})$. Given $\Theta$ the system faults can be reduced to $\mathcal{F} = \{F_{\text{hole}}, F_{\text{valve}}\}$ as described in Section 3.3, being the parents of the only observation $O_{\text{tire}}$. From $\mathcal{F}$ the new, reduced, observables $\mathcal{O} = \{O_{\text{tire}}, O_{\text{tube}}\}$ are obtained. The repair policy generator will be started with $\mathcal{F}_r = \mathcal{F}$, $O_c = (O_{\text{tire}})$ and the reduced Bayesian network as arguments. The result is shown in Table 4.2.

The domain of $\mathcal{F}$, $\Omega_\mathcal{F}$, is

$$\Omega_\mathcal{F} = \begin{pmatrix} \text{(inactive, inactive)} \\ \text{(inactive, active)} \\ \text{(active, inactive)} \\ \text{(active, active)} \end{pmatrix} = \begin{pmatrix} \varphi_0 \\ \varphi_1 \\ \varphi_2 \\ \varphi_3 \end{pmatrix}$$

Using Equation (3.6) the belief-state for $\mathcal{F}$ given the observation $O_{\text{tire}} = \text{flat}$ becomes

$$b = \begin{bmatrix} P(\varphi_0 | O_{\text{tire}} = \text{flat}) \\ P(\varphi_1 | O_{\text{tire}} = \text{flat}) \\ P(\varphi_2 | O_{\text{tire}} = \text{flat}) \\ P(\varphi_3 | O_{\text{tire}} = \text{flat}) \end{bmatrix} = \begin{bmatrix} P(O_{\text{tire}} = \text{flat} | \varphi_0) P(\varphi_0) \\ P(O_{\text{tire}} = \text{flat} | \varphi_1) P(\varphi_1) \\ P(O_{\text{tire}} = \text{flat} | \varphi_2) P(\varphi_2) \\ P(O_{\text{tire}} = \text{flat} | \varphi_3) P(\varphi_3) \end{bmatrix} = \begin{bmatrix} 0.0000 \\ 0.6667 \\ 0.1667 \\ 0.1667 \end{bmatrix}$$

Table 4.3 shows the expected cost for all policies given each $\varphi$ and the total expected cost.
| F. comb., ϕ | Observation | \( P(O_{\text{tire}} = \text{flat}|\varphi) \) |
|------------|-------------|----------------------------------|
| \{inactive, inactive\} | 0           |                                  |
| \{inactive, active\}   | 1            |                                  |
| \{active, inactive\}   | 1            |                                  |
| \{active, active\}     | 1            |                                  |

| F. comb., ϕ | Observation | \( P(O_{\text{tube}} = \text{visible hole}|\varphi) \) |
|------------|-------------|----------------------------------|
| \{inactive\} | 0           |                                  |
| \{active\}  | 0.6         |                                  |

| F. comb., ϕ | Observation | \( P(O_{\text{light}} = \text{no light showing}|\varphi) \) |
|------------|-------------|----------------------------------|
| \{inactive, inactive\} | 0           |                                  |
| \{inactive, active\}   | 1            |                                  |
| \{active, inactive\}   | 1            |                                  |
| \{active, active\}     | 1            |                                  |

| F. comb., ϕ | Observation | \( P(O_{\text{voltage}} = \text{voltage low}|\varphi) \) |
|------------|-------------|----------------------------------|
| \{inactive\} | 0           |                                  |
| \{active\}  | 1            |                                  |

Table 4.2: CPTs for the observations in Table 4.1.

<table>
<thead>
<tr>
<th>( \pi_i )</th>
<th>( C_{\pi_i}(\varphi_0) )</th>
<th>( C_{\pi_i}(\varphi_1) )</th>
<th>( C_{\pi_i}(\varphi_2) )</th>
<th>( C_{\pi_i}(\varphi_3) )</th>
<th>( C_{\pi_i}(b) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi_1 )</td>
<td>-</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>( \pi_2 )</td>
<td>-</td>
<td>140</td>
<td>170</td>
<td>170</td>
<td>165</td>
</tr>
<tr>
<td>( \pi_3 )</td>
<td>-</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>( \pi_4 )</td>
<td>- 170</td>
<td>70</td>
<td>170</td>
<td>103.3</td>
<td></td>
</tr>
<tr>
<td>( \pi_5 )</td>
<td>-</td>
<td>154</td>
<td>190</td>
<td>166</td>
<td>180</td>
</tr>
<tr>
<td>( \pi_6 )</td>
<td>-</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>( \pi_7 )</td>
<td>-</td>
<td>166</td>
<td>90</td>
<td>166</td>
<td>115.3</td>
</tr>
<tr>
<td>( \pi_8 )</td>
<td>-</td>
<td>156</td>
<td>150</td>
<td>174</td>
<td>155</td>
</tr>
<tr>
<td>( \pi_9 )</td>
<td>-</td>
<td>156</td>
<td>150</td>
<td>174</td>
<td>155</td>
</tr>
<tr>
<td>( \pi_{10} )</td>
<td>-</td>
<td>172</td>
<td>90</td>
<td>190</td>
<td>120.3</td>
</tr>
<tr>
<td>( \pi_{11} )</td>
<td>-</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>( \pi_{12} )</td>
<td>-</td>
<td>154</td>
<td>190</td>
<td>166</td>
<td>180</td>
</tr>
<tr>
<td>( \pi_{13} )</td>
<td>-</td>
<td>166</td>
<td>90</td>
<td>166</td>
<td>115.3</td>
</tr>
<tr>
<td>( \pi_{14} )</td>
<td>-</td>
<td>174</td>
<td>150</td>
<td>174</td>
<td>158</td>
</tr>
<tr>
<td>( \pi_{15} )</td>
<td>-</td>
<td>178</td>
<td>190</td>
<td>190</td>
<td>188</td>
</tr>
<tr>
<td>( \pi_{16} )</td>
<td>-</td>
<td>174</td>
<td>150</td>
<td>174</td>
<td>158.1</td>
</tr>
</tbody>
</table>

Table 4.3: The cost for all repair policies. The optimal repair policy is shown in bold font.
Figure 4.2: Repair policies generated for bicycle in Table 4.1. Branchings to the left correspond to a positive test result, i.e. that that observation is indicating a behavior that is not normal.
Chapter 5

Belief-State Search Tree

Our knowledge of the system is described completely by the belief-state and the decomposition state. As described in Section 3.4, for all fault revoking actions it is possible to calculate a new belief-state and for all testing actions 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 [6].

5.1 The AND/OR-Tree Representation

An AND/OR-tree 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 needs to be solved [4].

In the full search method, described in Chapter 4, repair policies were introduced. For this second method, where the solution is found by searching through the belief-state space, all possible repair policies are represented as a single AND/OR-tree. OR-nodes are junctions for alternative repair policies and AND-nodes are junctions for different test outcomes.

5.1.1 Edges and Nodes

The graphical representation of a repair policy has actions on the nodes and test outcomes on the edges to the children of a testing action node. In the AND/OR-tree the actions are placed on the edges from an OR-node to its children. 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 defines 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. In Figure 5.1 a simple AND/OR tree is shown.

The root node of the AND/OR-tree is always an OR-node with the initial belief-state and decomposition state. The labels of its child nodes are calculated from the actions on the connecting edges. Every OR-node has one child for each applicable action given the belief-state of the node. An applicable action is an...
action that affects the belief-state. Actions that do not affect the belief-state are pointless since they merely increase the the cost without contributing anything.

**Definition 14 (Applicable Action).** An action $a$ is an applicable action if the belief-state $b'$ after action $a$ is performed is not equal to the prior belief-state $b$:

$$b' \neq b$$

Every AND-node has one child for every possible outcome of the corresponding testing action. The children of an AND-node are always OR-nodes.

OR-nodes where the belief-state indicates that there is a sufficient probability that the system is free of faults are called goal nodes and they have no children. The level of probability that is regarded as sufficient is user defined.

### 5.1.2 Expected Cost of Repair

The calculation of the repair cost for the Belief-State Search Tree method is similar to that of the Full Search method, but differs slightly since the nodes now represent belief-states instead of actions. The cost of repair of a node $n$, $C(n)$, is the sum of the costs of all actions performed on a path from $n$ to a goal node. This cost is dependent on the choice of action at every OR-node and the test outcome at every AND-node. At OR-nodes the action deciding the successor node can be chosen freely, but at AND-nodes the outcome of the test, that is deciding the successor node, is stochastic.
For a goal node \( n \), the cost of repair is the cost of restoring the system to the decomposition state required by the goal criterion, \( c_{\text{restore}} \).

\[
C(n) = c_{\text{restore}} \quad (5.1)
\]

Since the cost is not stochastic the expected cost of repair becomes

\[
E\{C(n)\} = C(n) = c_{\text{restore}} \quad (5.2)
\]

The cost of repair for an OR-node \( n \) is the cost of repair of the chosen successor \( m \in ch(n) \) plus the cost of the corresponding action,

\[
C(n) = c(n, m) + C(m) \quad (5.3)
\]

where \( c(n, m) \) is the cost of the action on the edge between node \( n \) and \( m \), including a decomposition state transition if required. The expected cost of repair for an OR-node \( n \) is

\[
E\{C(n)\} = \sum_{m \in ch(n)} p(m)E\{C(m)\} \quad (5.6)
\]

The expected cost of an OR-node depends on the choice of action. The minimal expected cost is found if the child of every OR-node on a path is chosen to minimize the expected cost of repair. The minimal expected cost \( C_{\text{min}}(n) \) of repair of an OR-node \( n \):

\[
C_{\text{min}}(n) = \min_{m \in ch(n)} (c(n, m) + C_{\text{min}}(m)) \quad (5.7)
\]

If the next action at OR-nodes always is chosen to minimize the expected cost, the expected cost of repair of a node \( n \) is calculated as

\[
C_{\text{min}}(n) = \begin{cases} 
  c_{\text{restore}} & \text{if } n \text{ is an goal-node,} \\
  \min_{m \in ch(n)} (c(n, m) + C_{\text{min}}(m)) & \text{if } n \text{ is an OR-node,} \\
  \sum_{m \in ch(n)} p(m)C_{\text{min}}(m) & \text{if } n \text{ is an AND-node.}
\end{cases} \quad (5.8)
\]

From this it also follows that the minimal expected cost of repair for an entire tree is the minimal expected cost of the root node, \( C_{\text{min}}(\text{root}) \).
5.1.3 AND/OR-Tree for the Bicycle Example

In Figure 5.2 the complete AND/OR-tree is shown for the bicycle example with the same initial observations as in the example for the Full Search method. OR-nodes are represented with circles and AND-nodes with squares. The goal nodes are marked with stars.

Figure 5.2: The complete AND/OR-tree for the bicycle example.
5.2 Searching the AND/OR-Tree

Finding the minimum expected cost by searching the entire AND/OR-tree is fairly inefficient since the tree quickly becomes very large even for small problems. A depth-first search with branch-and-bound is a commonly used search strategy for algorithms for AND/OR-trees [3][8][16, p. 435]. With this search strategy a branch is searched until a goal node is reached and thereby an upper bound for the expected cost is found. The upper bound can then be used to prune\(^1\) other branches that end up costing more than best expected cost value so far. By using branch-and-bound large parts of the tree do not have to be searched. Going depth first rather than breadth first lets us find upper bounds faster thus allowing us to prune larger parts of the tree earlier.

5.2.1 The Search Algorithm

In order to use a branch-and-bound depth-first search algorithm we need to store more information on each node. We need to store the accumulated cost so far and an upper bound of the cost. The upper bound of a node is the highest allowed expected cost of reaching the goal state through this node. The upper bound is updated when the algorithm backtracks after a goal node is reached.

**Lower Bound**

We wish to prune those search branches that would end up costing more than the upper bound. To do that we want to have an estimated value of how much a certain path would cost at least. This is the lower bound of the node.

**Definition 15** (Lower bound). A function \(lb(n)\) of a node \(n\) can be used as a lower bound of a node \(n\) if and only if

\[
lb(n) \leq C_{\text{min}}(n)
\]

where \(C_{\text{min}}(n)\) is the minimum expected cost of reaching a goal state on the path passing through node \(n\).

In order to guarantee that no optimal solution is pruned, the lower bound has to be equal to or less than the cost of the optimal solution passing through the node. If the lower bound is a good estimate of the optimal solution more branches can be pruned at an earlier stage of the search, but a good estimate requires more calculations. A naive approach would be to let the lower bound be equal to the accumulated cost so far but this would result in searching many suboptimal branches almost to the end.

For our lower bound calculation we imagine a “perfect test” with zero cost that can determine exactly which fault combination the system has. For every fault combination we calculate the minimal cost of repairing only those faults. This test has the same amount of outcomes as there are fault combinations in the belief-state, each with the same probability as the probability to have that particular fault combination. Using Equation 5.8, the expected cost, if this test is made, becomes the accumulated cost so far plus the sum of the cost

---

\(^1\)Pruning is a term in informatics for cutting of parts of a search tree that do not contain the object being searched for
of repairing each fault combination weighted with its probability. The cost of repairing each fault combination is dependent on the decomposition state. To simplify, and to guarantee the lower bound to always be less than the optimal solution, the cost of transitions in the decomposition state are ignored. Finding this cost is trivial if there is only one fault revoking action that repairs each fault. If several actions can revoke the same fault the optimal choice of actions revoking all faults in the fault combination has to be chosen. Even if this might take a little bit more time the results can be reused in other nodes.

Since the “perfect test” gains full information and has zero cost no other test can outperform it. This means that if this cost is used as a lower bound for a node it is always equal to or less than the optimal solution that can be found from this node. Let $C_{\min}(\varphi)$ be the minimum cost of repairing the faults in the fault combination $\varphi$, then the equation for the lower bound of an OR-node $n$, $lb(n)$, is

$$lb(n) = c(n) + \sum_{\varphi \in \Omega_F} C_{\min}(\varphi)b(\varphi)$$  \hspace{1cm} (5.9)$$

where $c(n)$ is the accumulated cost so far in the node and $b$ is the belief-state of the node.

AND-nodes do not contain belief-states, but since the test resulting in the AND-node always will be outperformed by the “perfect test”, the test only contributes with an extra cost. Therefore, the cost of performing the previous testing action plus the lower bound of the parent node is a valid lower bound of an AND-node. The equation for the lower bound of an AND-node $n$ is

$$lb(n) = c_a + lb(pa(n))$$  \hspace{1cm} (5.10)$$

where $c_a$ is the cost of performing the previous action and $pa(n)$ is the parent node.

For goal-nodes the lower bound does not need to be an estimate since no further actions are needed. It is calculated as the accumulated cost so far plus the cost of restoring the decomposition state $c_{restore}$:

$$lb(n) = c(n) + c_{restore}$$  \hspace{1cm} (5.11)$$

Expanding Nodes

When a node is searched it has to be expanded to create the children of the node. For an OR-node, first all applicable actions are determined, then corresponding nodes are created for each action, OR-nodes for fault revoking actions and AND-nodes for testing actions. Children with a higher lower bound than the upper bound of the parent are immediately pruned. When the children of an OR-node are searched they inherit the upper bound of the parent. The upper bound of the parent is updated whenever a cheaper solution than the previous upper bound is found.

When an AND-node is expanded one child is created for each test outcome. To determine the cost of a path passing through an AND-node all children have to be searched. The upper bound of each child node has to be such that if the cost of a path passing the child node exceeds the upper bound of the child node the upper bound of the parent node is exceeded too. The upper bound of a child node $m$ to the AND-node node $n$, $ub(m)$, is calculated according to Theorem 1.
**Theorem 1** (Upper Bound of the Child to an AND-Node). Let \( n \) be an AND-node with its children contained in the set \( \text{ch}(n) \) and \( \forall m \in \text{ch}(n), \, p(m) > 0 \). Further, let the upper bound \( \text{ub}(m) \) of a node \( m \in \text{ch}(n) \) be calculated as

\[
\text{ub}(m) = \frac{\text{ub}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{lb}(m')}{p(m)} \quad (5.12)
\]

Then it holds that if \( \exists m \in \text{ch}(n) \) such that

\[
\text{C}_{\text{min}}(m) > \text{ub}(m) \quad (5.13)
\]

then

\[
\text{C}_{\text{min}}(n) > \text{ub}(n) \quad (5.14)
\]

**Proof.** When \( n \) is an AND-node (5.8) can be rearranged to

\[
\text{C}_{\text{min}}(m) = \text{C}_{\text{min}}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{C}_{\text{min}}(m') \quad (5.15)
\]

Inserting (5.15) and (5.12) in (5.13) yields

\[
\text{C}_{\text{min}}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{C}_{\text{min}}(m') > \text{ub}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{lb}(m') \quad (5.16)
\]

\[
\Leftrightarrow \quad \text{C}_{\text{min}}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{C}_{\text{min}}(m') > \text{ub}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{lb}(m') \quad (5.17)
\]

Since \( \text{lb}(n) \leq \text{C}_{\text{min}}(n) \) according to Definition 15

\[
\text{C}_{\text{min}}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{lb}(m') \geq \text{C}_{\text{min}}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{C}_{\text{min}}(m') \quad (5.18)
\]

Inserting (5.18) in (5.17) yields

\[
\text{C}_{\text{min}}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{lb}(m') > \text{ub}(n) - \sum_{m' \in \text{ch}(n)} p(m') \text{lb}(m') \quad (5.19)
\]

Since \( p(m) > 0 \)

\[
\text{C}_{\text{min}}(n) > \text{ub}(n) \quad (5.20)
\]

\[ \square \]

**The Algorithm**

Pseudo-code for the algorithm called R-BBDFS (Recursive Branch-and-Bound Depth First Search) is shown in Algorithm 6. The function \( \text{search}(n) \) is called to search a node \( n \) and returns a minimum expected cost \( c \). It expands nodes by
calling the function $\text{expand}(n)$ that returns a sorted list containing the expanded children of the node $n$.

**function** search($n$)
**returns** expected minimal cost of repair, $c$

switch do
| case $n$ is a Goal-node |
| $c = \text{lb}(n)$ |
| case $n$ is an OR-node |
| $ch(n) = \text{expand}(n)$ |
| forall $m \in ch(n) : \text{lb}(m) < \text{ub}(n)$ do |
| $ub(m) = \text{ub}(n)$ |
| $c_m = \text{search}(m)$ |
| if $c_m < \text{ub}(n)$ then |
| $\text{ub}(n) = c_m$ |
| best = $m$ |
| end |
| end |
| $ch(n) = \{\text{best}\}$ |
| $c = \text{ub}(n)$ |
| case $n$ is an AND-node |
| $ch(n) = \text{expand}(n)$ |
| forall $m \in ch(n)$ do |
| if $\text{lb}(n) < \text{ub}(n)$ then |
| $\text{lb}(n) = \frac{\text{lb}(n) - p(m)\text{lb}(m)}{p(m)}$ |
| $ub(m) = \frac{\text{ub}(n) - \text{lb}(n)}{p(m)}$ |
| $c_m = \text{search}(m)$ |
| if $c_m < \text{ub}(m)$ then $\text{lb}(n) = \text{lb}(n) + p(m)c_m$ |
| else $\text{lb}(n) = \infty$ |
| end |
| end |
| $c = \text{lb}(n)$ |
end

end
**return** $c$

**Algorithm 6: R-BBDFS**

### 5.2.2 Heuristics

The computational complexity of the search is proportional to the amount of nodes expanded which is $O(f^d)$ where $f$ is the mean branching factor and $d$ is the depth of the search if the time to calculate the labels of each node is constant. The complexity of the search problem can be reduced with the use of heuristics.

**Sorting the Children of an OR-Node**

After a node has been expanded, the child node to be searched can be chosen better or worse. A simple way to speed up the search is to order the children in the order they are to be searched so that the optimal solution is found earlier.
and larger parts of the total search tree are pruned. An exception is made for goal-nodes which are always ordered first since they are quickly searched.

When searching the children of an OR-node, finding a small solution early reduces the need of searching the other nodes extensively. The idea here is to order the nodes ascending by some heuristic \( h(n) \) that estimates the minimum expected cost in a node \( n \).

The lower bound is already used in the algorithm, so by letting \( h(n) = lb(n) \) no extra calculations are needed. Later on we will compare the different heuristics so we call this heuristic \( h_1 \). However, with \( h_1 \) a cheap test is always favored over an expensive one since the lower bound is the expected cost of repair under the condition that information of the state of the system is free.

To handle this, we could add to the previous heuristic, a cost of the remaining entropy of the belief-state in the node. In information theory [7, p. 18], the entropy of a stochastic variable \( X \), \( H(X) \), is calculated as

\[
H(X) = - \sum_{x \in X} P(x) \log P(x)
\]  

Since OR-nodes contain a belief-state Equation 5.21 can be used directly to calculate the entropy. In an AND-node there is no belief-state, so the belief-states of its children are used instead (this requires that AND-nodes are expanded immediately upon creation). In AND-nodes the conditional entropy [7, p. 42] is used, i.e. the entropy of a stochastic variable given that the value of a second variable is known. The conditional entropy of a stochastic variable \( X \) given that the value of a second variable \( Y \) is known, \( H(X|Y) \), is

\[
H(X|Y) = - \sum_{y \in Y} P(y) \sum_{x \in X} P(x|y) \log P(x|y)
\]

For a node \( n \) the remaining entropy \( H(n) \) is calculated as

\[
H(n) = \begin{cases} 
- \sum_{\varphi \in \Omega_x} b_n(\varphi) \log b_n(\varphi) & \text{if } n \text{ is an OR-node} \\
- \sum_{m \in ch(n)} p(m) \sum_{\varphi \in \Omega_x} b_m(\varphi) \log b_m(\varphi) & \text{if } n \text{ is an AND-node}
\end{cases}
\]

where \( b_i \) is the belief-state of node \( i \).

The heuristic \( h_2 \) using entropy is calculated as

\[
h_2(n) = lb(n) + c_{\text{entropy}} H(n)
\]

where \( c_{\text{entropy}} \) is a constant designating the mean cost of reducing entropy. This cost can be estimated experimentally using training data with the least-squares method,

\[
c_{\text{entropy}} = (H^T H)^{-1} H^T (lb - C_{\text{min}})
\]

where \( H \), \( lb \), and \( C_{\text{min}} \) are column vectors containing measured values of entropy, lower bound, and minimum expected cost for all nodes in the training data.

**Sorting the Children of an AND-Node**

Since all children of an AND-node have to be searched there is no gain in searching paths that contain cheaper solutions first. Here the strategy in choosing the
next search branch is by choosing the most disproving search branch first [17]. According to Theorem 1 the upper bound of the AND-node is exceeded if any child exceeds the upper bound for its branch. This means that searching children that are likely to exceed their upper bound first decreases the chance of needing to search all of the remaining children. The children are ordered ascending by the difference between their upper bound and their lower bound. For a node $m$ with the parent $n$ this difference, denoted by $D(m)$, becomes

\[
D(m) = ub(m) - lb(m) = \frac{ub(n) - \sum_{m' \in \text{ch}(n)} p(m')lb(m')}{p(m)} - \frac{p(m)lb(m)}{p(m)} = (5.26)
\]

Since $lb(n) - \sum_{m' \in \text{ch}(n)} p(m')lb(m')$ is the same for all children the child with the greatest probability $p(m)$ is also the child with the lowest difference $D(m)$.

**Limiting Search Depth and Branching Factor**

So far, the optimal solution is guaranteed to be found using the above mentioned heuristics, but the worst case computation time is still exponential. An easy way to make the algorithm perform in polynomial time is to limit the recursion depth. When the recursion limit is reached the cost of the remaining path is approximated by the minimum cost of repairing all faults that have a probability greater than zero using only fault revoking actions. If $d$ is the recursion depth limit and $d^*$ is the depth of the shallowest optimal solution the optimal solution will always be found if $d^* \leq d$. The worst case computational complexity of the problem is still $O(f^d)$ but now $d$ is a constant.

The computational complexity of the search problem can be reduced further if an upper limit to the branching factor is set. This is done by only allowing the algorithm to search a certain number of the branches. The remaining branches are pruned. If the heuristics sorting the children of expanded nodes succeed in sorting the node containing the optimal path among those nodes that are not cut off by the branching factor limit, the optimal solution will still be found.

By limiting the search depth an optimal solution will never be found if the depth of the shallowest optimal solution is greater than the limit, even if it would have been found quickly with a higher or no depth limit. This can be handled by using iterative deepening [16, p. 78]. The algorithm is started with a small recursion depth limit and once a solution is found the algorithm restarts with an increased depth limit. After a certain time the search is aborted and the best solution found so far is returned. It might seem time consuming to research the tree many times but the time complexity does not increase since

\[
O(f^1 + f^2 + \ldots + f^n) = O\left(f^n \frac{1 - f^{-n}}{1 - f^{-1}}\right) = O(f^n) \quad (5.27)
\]
5.2.3 Demonstration of Search Algorithm on the Bicycle Example

To demonstrate this search algorithm we apply it on the bicycle example. For this example the children of OR-nodes are sorted according to the heuristic $h_1$ and the children of AND-nodes are sorted descending by their probabilities. There are no limits of the recursion depth and the branching factor and the decomposition state is ignored. In Figure 5.3 the complete AND/OR-tree for the example is shown and we can follow how the algorithm searches the tree to find the optimal solution.

Before the algorithm is started the first node is created. The variables $lb$ and $c$ represent the lower bound of the node and the accumulated cost so far in the node. The action that is performed to reach the current node is indicated in the field previous action ($pa$). The minimum cost to repair the faults of a certain combination of faults $\varphi \in \Omega_F$ is represented in the column $C_{min}(\varphi)$. The contents of some of the nodes are displayed in full in the small tables. To simplify reading not all expanded nodes are displayed.

The algorithm starts by expanding node 1. The applicable actions are $r_{tube}$, $r_{valve}$ and $t_{tube}$. Three nodes are created:

<table>
<thead>
<tr>
<th>Node: 1</th>
<th>$F = {F_{hole}, F_{valve}}$</th>
<th>$\Theta = {O_{tire}}$</th>
<th>$p$</th>
<th>$b(\varphi)$</th>
<th>$C_{min}(\varphi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: OR</td>
<td>$\varphi$</td>
<td>$b(\varphi)$</td>
<td>$C_{min}(\varphi)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$lb$</td>
<td>58.33</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>{⊥, ⊥}</td>
<td>0.1667</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>$pa$</td>
<td>-</td>
<td>{⊥, ⊤}</td>
<td>0.6667</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{⊤, ⊥}</td>
<td>0.1667</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{⊤, ⊤}</td>
<td>0.1667</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

All new nodes have lower bounds less than the upper bound ($\infty$). In node 4 the values for $P(\theta|b)$ are calculated as $1 \cdot 0 + 1 \cdot 0.6667 + 0.4 \cdot 0.1667 + 0.4 \cdot 0.1667 = 0.8$ and $0 \cdot 0 + 0 \cdot 0.6667 + 0.6 \cdot 0.1667 + 0.6 \cdot 0.1667 = 0.2$. Since node 3 has the lowest lower bound it is searched first. It is expanded into three new nodes. One of them, after $r_{tube}$ is performed at a cost of 100, is a goal node so the upper bound becomes 130. The other two nodes are the following:

<table>
<thead>
<tr>
<th>Node: 2</th>
<th>$F = {F_{valve}}$</th>
<th>$\Theta = \emptyset$</th>
<th>$p$</th>
<th>$b(\varphi)$</th>
<th>$C_{min}(\varphi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: OR</td>
<td>$\varphi$</td>
<td>$b(\varphi)$</td>
<td>$C_{min}(\varphi)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$lb$</td>
<td>125</td>
<td>1.6667</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>{⊥}</td>
<td>0.8333</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node: 3</th>
<th>$F = {F_{hole}}$</th>
<th>$\Theta = \emptyset$</th>
<th>$p$</th>
<th>$b(\varphi)$</th>
<th>$C_{min}(\varphi)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: OR</td>
<td>$\varphi$</td>
<td>$b(\varphi)$</td>
<td>$C_{min}(\varphi)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$lb$</td>
<td>63.33</td>
<td>0.6667</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>{⊥}</td>
<td>0.3333</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

| Node: 4 | $F = \{F_{hole}, F_{valve}\}$ | $\Theta = \{O_{tire}, O_{tube}\}$ | $p$ | $b(\theta|b)$ | $P(\theta|b)$ |
|---------|-------------------------------|--------------------------------|-----|-------------|------------------|
| type: AND | $\varphi$ | $b(\theta|b)$ | $P(\theta|b)$ |
| $lb$ | 78.33 | 0.8 | 0.2 |
| $c$ | 20 | {⊥} | 0.8 |

pa $t_{tube}$ | {⊤} | 0.2 |
Node 6 has the lowest lower bound of these two and is searched first. It has two children, one for each outcome, with lower bounds 66.67 and 150 respectively. From here on the contents of the nodes will not be displayed in full. The first child has the highest probability and is searched first. The upper bound of this node is \((130 - 0.2 \cdot 150)/0.8 = 125\). In this node there is only one available action, \(t_{tire}\), since the lower bound of the node created by performing \(r_{tube}\) is 150 which is greater than 125. The node created by performing \(t_{tire}\) has a lower bound of 106.67 which is less than 125. This node is expanded further and a solution is found with the cost 106.67. The second child of node 6 must also be expanded since node 6 is an AND-node. The upper bound becomes \((130 - 0.8 \cdot 106.67)/0.2 = 223.33\) which is greater than the lower bound 150, so it is expanded. A solution for this child is found to be 150. So the expected cost of the solution in node 6 becomes \(0.8 \cdot 106.67 + 0.2 \cdot 150 = 115.33\).

When node 5 is searched the upper bound is 115.33. The solution found there turns out to have an expected cost of 103.33 which is an improvement so the upper bound in node 1 is updated.

Next in line to be searched is node 4 since it has a lower bound of 78.33. The child with the highest probability has the upper bound \((103.33 - 0.2 \cdot 135)/0.8 = 95.42\). When it is searched, no solution is found with a cost less than the upper bound. This means that node 4 can be discarded.

Node 2 already has a lower bound greater than the upper bound so it can also be discarded. Since all previous actions are stored in the nodes the optimal repair policy is easy to extract. The optimal repair policy is:

\[
\begin{align*}
&\text{value} \\
&\downarrow \\
&t_{tire} \\
&\downarrow \\
&r_{tube} \\
&\star \\
&\downarrow \\
&\star
\end{align*}
\]

and it has an expected cost of 103.33.
Figure 5.3: AND/OR search-tree for the bicycle example. The numbering shows the order in which the nodes are searched. The parts of the tree shaded in gray are never searched by the algorithm and the parts shown with thick lines show the optimal solution.
5.3 Search with Decomposition State

In this section we will describe how the search is affected by including the decomposition state and how it affects the calculation of the lower bound.

The decomposition state affects the costs when actions are performed that require transitions in the decomposition state. In Section 5.2.1 the costs of transitions in decomposition state were ignored when calculating the lower bound. If the costs of transitions are relatively high compared to the cost of performing actions the lower bound estimation can become too optimistic to be efficient.

Ignoring the decomposition state offers the possibility to precalculate the lower bounds for each combination of faults. Calculating a better lower bound, including the costs of transitions, comes with a larger computational overhead since these costs depend on the decomposition state in the node. Apart from storing the minimum cost of repairing each combination of faults, information of which decomposables need to be decomposed at least once when repairing the corresponding combination of faults needs to be stored. When calculating the lower bound of a node the cost of decomposing these decomposables $c_{\text{decompose}}$ is added to the cost of repairing the faults. Also added is the cost of restoring the system to a state where all decomposables are assembled, $c_{\text{restore}}$. Using this as a lower bound for a node still guarantees it to be less than or equal to the minimum expected cost of the optimal solution in the node. The value $c_{\text{decompose}}$ is calculated by applying Algorithm 4 once for every decomposable that needs to be decomposed at least once. On the first call of Algorithm 4 the input decomposition state is the decomposition of the node. On the consecutive calls the input decomposition state is the output of the previous call. $c_{\text{restore}}$ is calculated in the same way using Algorithm 3 where the first input decomposition state is the output decomposition state of the last call of Algorithm 4. The equation for calculating the lower bound, Equation 5.9, is replaced by

$$\text{lb}(n) = c(n) + \sum_{\varphi \in \Omega_F} (C_{\text{min}}(\varphi) + c_{\text{decompose}} + c_{\text{restore}}) b(\varphi)$$

(5.28)
Chapter 6

Some Existing Methods

In this chapter we present two existing methods for solving planning problems under partial observability, discuss strengths and weaknesses, and point out the most important differences between the approaches. The first method is referred to as Heckerman’s method, and has been subject to earlier master theses at Scania. To this method two implementations are presented, meaning that we have the possibility to compare the performance of these implementations on test cases with the performance of the Belief-state Search Tree method. The second one is called Partial Observable Markov Decision Process (POMDP), an extension of the Markov Decision Process.

6.1 Heckerman’s Method

In [9], Heckerman, Breese and Rommelse present a method for decision-theoretic troubleshooting, and proves it to be optimal under certain assumptions. Then some of the assumptions are relaxed, with loss of optimality but with satisfactory empirical results. This section is a partial description of that method. For a full description we refer to the article.

The system model is a Bayesian network, and for our discussion the most important assumptions are:

1. Exactly one component is abnormal and is responsible for the failure of the system. This also implies that the system is faulty at the beginning of the troubleshooting.

2. Directly after any component repair, the system functionality is observed with cost $C_p$.

3. Each component is observable or unobservable. An observable component can be tested or inspected with full certainty to determine if it is functioning properly or not. An observable component that is observed to be abnormal must be repaired immediately. An unobservable component can never be directly observed, but can be repaired or replaced.

4. The costs of observation and repair of any component do not depend on previous repair or observation actions.
5. No observations other than those in the assumptions above are available.

Now assume that the system has \( n \) components represented by variables \( c_1, \ldots, c_n \), and that each component is in exactly one of a finite set of states. Initially only observable components are regarded. The cost of observation and the cost of repair of component \( c_i \) are denoted \( C_i^o \) and \( C_i^r \) respectively. The probability that component \( c_i \) is causing the failure of the system is denoted by \( p_i \). If the components are observed and possibly repaired in the order \( c_1, \ldots, c_n \), then the expected cost of repair, here denoted \( ECR(c_1, \ldots, c_n) \), is

\[
ECR(c_1, \ldots, c_n) = (C_1^o + p_1(C_1^r + C^p)) + (1 - p_1)(C_2^o + \frac{p_2}{1 - p_1}(C_2^r + C^p)) + \ldots + (1 - p_1 - p_2) \ldots (n-1) \ldots (1 - p_1 - \ldots - p_{i-1})\left[ C_i^o + p_i(C_i^r + C^p) \right] \quad (6.1)
\]

This equation put into words would be: first \( c_1 \) is observed, with a cost \( C_1^o \). This component is found to be faulty with probability \( p_1 \), and will then be repaired and the system functionality will be observed with the cost \( C_1^r + C^p \). In accordance with assumption 1 the system must now be functioning and the troubleshooting can end, since the only fault has been repaired. With the remaining probability \( 1 - p_1 \) it is found that \( c_1 \) is functioning, and then \( c_2 \) is observed. With probability \( p_2/(1 - p_1) \) (i.e. the probability that \( c_2 \) is faulty, if \( c_1 \) is known to be functioning) it is found that \( c_2 \) is faulty and shall be repaired, etc.

In the article it is proved that the optimal sequence of observation and repair is obtained by ordering the components by the ratio \( p_i/C_i^o \), called efficiency. The component with the highest efficiency is observed first, which is intuitively reasonable.

The algorithm for troubleshooting is now simple:

1. Compute the probabilities \( p_i \) that the component is faulty, given that the system is not functioning, using Bayesian network calculations.
2. Observe the component with the highest ratio \( p_i/C_i^o \) (that was not yet observed).
3. Replace the component if it is faulty.
4. End troubleshooting if a component was replaced, otherwise go to step 2.

Unobservable components can easily be included, but instead of observing the components they are replaced/repaired, and then assumed to be functioning. This is done by setting \( C_i^o \) to a repair cost \( R_i \) for those components, and then letting \( C_i^r \) be set to zero.

One advantage with this method is that the probabilities \( p_i \) only have to be calculated once, in step 1 of the algorithm.
6.1.1 Multiple Faults

Now, to adapt the method to realistic cases and to make it more general, some of the assumptions are relaxed. Multiple faults are allowed (relaxing assumption 1), with the consequence that the probabilities $p_i$ have to be recalculated every time a component is replaced. The new algorithm is:

1. Compute the probabilities $p_i$ that the component is faulty, given the current state of information.
2. Observe the component with the highest ratio $p_i/C_i$ (that was not yet observed).
3. Replace the component if it is faulty.
4. End troubleshooting if a component was replaced and if the system after the replacement is observed to be functioning, otherwise go to step 1.

The differences here lay in step 1 and 4. In step 4, the troubleshooting is only ended if the system is observed to be functioning, since finding and replacing a faulty component no longer guarantees that all faults have been handled. This version is computationally more demanding, since the probabilities are recalculated in step 1, using all new information.

Optimality is lost since there may exist cases where it would not be optimal to immediately repair a component, even though it was found to be faulty. The authors argue that since it mostly makes sense to repair a component immediately if it is found to be faulty, the loss of optimality is acceptable.

6.1.2 Introducing More Observations

Assumption 5 is partially relaxed. The two types of observations already introduced, i.e. observing whether the system is functioning or not, and observing the functionality of the individual components, are referred to as base observations. In this relaxation, it is assumed that at most one non base observation can be made initially, and after that the troubleshooting would go on as described above, with a sequence of repairing actions and base observations. Such a sequence is generated using the method in Sections 6.1 and 6.1.1. Let $ECR(I)$ denote the expected cost of such a sequence, generated with the current state of information $I$. Let $o_b$ be a non base observation that can be made with a cost $C_o$, and $o_b$ have $r_b$ possible outcomes. The event that $o_b$ takes on the state $k$ is denoted by $o_b = k$. The expected cost of repair with observation, $ECO$, is calculated as

$$ECO(I, o_b) = C_o + \sum_{k=1}^{r_b} P(o_b = k) ECR(I \cup \{o_b = k\})$$  \hspace{1cm} (6.2)

Note that the troubleshooting sequence following the observation may be different for every possible outcome of the observation.

When deciding whether or not to make a non base observation, the expected cost of repair with an observation, $ECO(I, o_b)$, is compared with the $ECR(I)$ for the same state of information, for all non base observations $o_b$. The action to carry out is the one corresponding to the lowest expected cost of repair or expected cost of repair with observation.
6.1.3 The LMS Method

In [13], [14] and [18] Lotz, Mossberg and Sundberg apply a variant of Heckerman’s method on the injection system of a truck. We call this method the LMS method. The system is called XPI (Extra High Pressure Injection), and is a Scania part. Assumption 5 is relaxed as above, using the notation test for a non base observation. Assumption 1 is not relaxed, i.e. the single fault assumption is still valid.

Further, a method for relaxing Assumption 4 is developed, with the argument that independent costs would not be realistic for such a system. An Assembly Level (AL) is introduced, describing to which degree the system is decomposed. The cost of observation or repair is said to possibly vary a lot depending on that level, e.g. on whether the truck is complete or if the cab is tilted. From the AL each observing, repairing and testing action is assigned a corresponding level, a level of observation, level of repair or level of test. Changing level from \( l_i \) to \( l_j \) has a certain cost \( C(l_i \rightarrow l_j) \).

When the assumption about independent costs is relaxed, the cost of an action in Equation 6.1 will depend on the AL for the previous action made. The new cost for an action, with AL, is derived from the original cost the action had before AL was introduced, with the new cost being

\[
C_{\text{new}} = C_{\text{original}} + C(l_i \rightarrow l_j),
\]

where \( l_i \) is the AL of the system before the action is performed, and \( l_j \) the AL assigned to the action.

The sequence, in which the components shall be observed and possibly repaired, can now be obtained through the following algorithm:

1. Calculate the cost \( C_i^o \) for observing each component, starting from an initial AL \( l_0 \).
2. Choose the component with the highest ratio \( p_i / C_i^o \) as the next component of the sequence.
3. Calculate the cost for observing each of the remaining components, starting from the AL of the previous action.
4. Go to step 2 if there are any components that have not yet been added to the repair sequence, otherwise terminate.

The components are observed and possibly repaired according to the obtained sequence. If a component is found to be faulty, the component is repaired, followed by a system functionality test. The repair process is ended when such a test finds the system to be functioning. The LMS method will be compared to the Belief-State Search Tree method in Section 8.2.

6.1.4 Dezide Advisor

In a joint work between the Research Unit for Decision Support Systems at Aalborg University and Customer Support R&D at Hewlett-Packard a troubleshooting system was developed, following the principles of Heckerman’s method [12] [10]. The method is partly implemented in the troubleshooting tool.
Advisor that is available from the Danish company Dezide, focusing on automation of customer service in telecoms [12] [5]. The tool is created to offer printer users a web-based interface to a decision-theoretic troubleshooter system, it is not intended exclusively for maintenance personnel who are trained to handle the equipment that is to be repaired. In Section 8.2 the performance of this tool is compared with the performance of the Belief-state Search Tree method.

6.2 POMDPs

POMDPs are Partially Observable Markov Decision Processes that can be used to solve decision problems in partially observable domains. The theory mentioned in this section is recited from an article by Cassandra, Kaelbling and Littman [1] describing the fundamentals of POMDP theory.

The POMDP framework is an extension of the normal Markov Decision Process framework (MDP) where the probability of being in a certain state at time step \( t + 1 \) only depends on the state and the performed action at time step \( t \). This dependence on only the previous time step is known as the Markov Property. When solving an MDP it is sufficient to find the optimal action to perform, for every possible state of the system. The optimal action is the action that minimizes some cost function, or maximizes some reward or value function. This mapping of states to actions is called a policy.

For POMDPs the current state of the system is not known. Instead, based on the previous observations, a probability distribution over all possible states, a belief-state, is calculated. The observations also follow the Markov property, i.e. the probability of making a certain observation is only depending on the previous state and the previous action. Given a correctly calculated belief-state no additional data of past actions and observations would supply any further knowledge about the state of the system. Therefore the optimal action to perform only depends on the current belief-state. This is just like for the MDP but instead of having a discrete state space the state space is now continuous. With the continuous state space it is no longer possible to find an optimal policy for every state since there are an infinite number of belief-states.

For POMDPs the policies are replaced by policy trees. A policy tree is a plan of what actions to take, depending on the outcome of the observations that are made after each action is performed, see Figure 6.1. For every state \( s \) it is possible to calculate the expected value \( V_p(s) \) of a policy tree \( p \). The expected value of executing a policy tree from some belief-state \( b \) is

\[
V_p(b) = \sum_{s \in S} b(s)V_p(s)
\]  

(6.4)

where \( b(s) \) is the probability of being in state \( s \) and \( S \) is the set of all states. The optimal policy tree \( p^* \) is found as

\[
p^* = \arg \max_{p \in P} b \cdot \alpha_p
\]  

(6.5)

where \( \alpha_p = \langle V_p(s_1), \ldots, V_p(s_n) \rangle \) and \( P \) is the set of all policy trees. Instead of finding the policy tree that maximizes the value it is possible, as in our case, to try to find the policy tree that minimizes the cost. Once all the values of \( V_p(s) \) are known Equation 6.5 can be solved with simple optimization methods.
This property can make it simple to solve POMDPs online\textsuperscript{1} since the values of $V_p(s)$ can be precalculated offline for all $p \in \mathcal{P}$ and $s \in \mathcal{S}$. The policy trees can then be mapped to the areas of the belief-state space where they are optimal to perform.

For problems with a large state space there are many possible actions and observations, meaning that $\mathcal{P}$ becomes very large. With finite time horizon $t$ the total number of policy trees can be as many as

$$|\mathcal{A}|^{|\mathcal{O}|^{t-1}}$$

where $|\mathcal{A}|$ is the number of possible actions and $|\mathcal{O}|$ is the number of possible observations. Even though there are good methods of solving POMDPs, problems with more than a few dozen states are often infeasible [16, p. 625–628].

6.3 Problematic Properties of Existing Methods for the Particular Problem Formulation

Regarding the fact that there exist methods for handling troubleshooting problems, is it then motivated to develop the method in this report? There are several properties of the methods described in this chapter, that make them unsuitable for troubleshooting a truck.

\textsuperscript{1}Online here meaning calculations that are made during run time. Offline calculations are pre-made, and are read from the memory during run time. Offline calculations can reduce run time, but demands more memory.
6.3.1 Heckerman

A main advantage of Heckerman’s method is that it does not explicitly construct a decision tree. This demands less calculations than if decision trees are constructed, and therefore makes the method very fast. It is a consequence of the merely partial relaxation of Assumption 5, only allowing observations as the first action, followed by a sequence containing only repair actions. Not allowing observations further on in the sequence is called a myopic (short sighted) method. It does not give an optimal expected cost of repair, but could still be sufficient to obtain the correct suggested action.

However, there are some crucial disadvantages of Heckerman’s method. Primarily, it is the observation of the system functionality, following Assumption 2, that is not applicable in the case of troubleshooting a truck. If this observation cost is high in relation to other repair costs, which often is the case when repairing a truck, the total cost of the troubleshooting will increase rapidly if there are many components that are repaired before the system is found to be fault free.

Neither is Assumption 4 possible in our case. In the LMS method, the dependency between costs and decomposition level is included, but since the system functionality is observed after each repair action, the full advantage of introducing dependent costs can not be taken.

6.3.2 POMDP

In the problem formulation in Section 1.2 it is stated that the problem is characterized by having a large state space, but a small support. It also states that the aim is to reach a goal state where the system is free of faults. If the policy trees were to be precalculated offline they would have to be able to reach the goal state from every possible initial state. This can lead to an intractable number of policy trees. The small support results in that every instance of the problem has a rather small state space, and only a fraction of the possible policy trees would need to be considered.

Since actions that make observations always yield the same result unless the system is altered, tests never need to be performed several times in a row. Actions that revoke a fault always succeed which means that they also do not need to be performed twice. This means that there is a finite number of belief-states that can be reached from any given belief-state. With a finite amount of reachable belief-states the problem can be solved by searching the belief-state space as proposed in Chapter 5 and there is no need to try to map the policy trees to the belief-state space. If the mapping is to be made offline the policy trees would have to be mapped to the belief-state space of the entire state space. With a large state space of thousands of possible faults this mapping can lead to infeasible memory requirements.

As a conclusion, there is motivation to look for new methods of troubleshooting and planning, since the area of automated troubleshooting of today is not fully applicable to our problem formulation.
Chapter 7

Modeling of the Test Case

In a master thesis written at Scania by Mossberg [14] the XPI-system is modeled as a Bayesian network, used for a troubleshooting problem. Since creating a complete probabilistic model is time consuming, it is convenient to use the already existing model to evaluate the Belief-State Search Tree method. Further, in a master thesis at Scania written by Sundberg [18], a series of troubleshooting simulations are made. The simulations use the implementation of Heckerman’s method as described in Section 6.1.3 on the XPI-model. By using the same model and the same test case input data as Sundberg, we can compare the performance of our method with the performance of Heckerman’s method.

7.1 The XPI-model

This is a brief description of the XPI-model as it is implemented in the previous master theses, using the notation used in the Sections 6.1 and 6.1.3. For a more detailed description of the XPI-system and how it is modeled, it is recommended to read [13], [14], [18].

The model of the XPI-system has 17 components that can be faulty. There are nine so-called DTCs (Diagnostic Trouble Codes). In reality, these are trouble codes generated in the vehicle computer, to aid the mechanic in the troubleshooting. Here they are modeled as observations that are read at the beginning of the repair process, thus affecting the probabilities for the different faults, before any decisions about repair or testing actions have been made. Seven of the components are observable. This means that there exists a test that can determine if a component is faulty or not with 100% certainty. Apart from observing the components directly there are 13 tests that can be made.

For each component there is a repair action and for each observable component there is an additional observation action. The tests are implemented as test actions.

There is also a model of the decomposition state for the XPI-system with eight different states. Every repair and observation action has a required level of repair and a level of observation and every testing action has a required level of test. Transitions between the different levels of the decomposition state are associated with a certain cost.

Mossberg’s Bayesian network model has three levels of nodes. Two of the
levels are very similar to the models in this report (Section 3.2). One level has nodes representing the components, and a second layer has nodes representing the DTCs and the tests. In Heckerman’s method a single fault assumption is made. To implement this assumption in the model, Mossberg has added a control node above the component layer, ensuring that only one component can be faulty at a time. In the CPT for the control node are the a priori probabilities for each component to be faulty.

The control node has one connection to each component node, i.e. the probabilities for the components to be faulty are conditional on the mode of the control node. The probabilities for the DTCs and the tests are conditional on some or several components, as seen in Figure 7.1.

Figure 7.1: The XPI-system modeled as a Bayesian network. The control node is labeled SFC, the components Qᵢ, the tests Tᵢ and the DTCs are labeled Dᵢ.

7.2 Adaption of the XPI-model

In order to be able to use Mossberg’s XPI-model with the framework in this thesis, certain adaptions and changes have to be made.

The components are interpreted as faults in our framework and the DTCs and tests are interpreted as observables. The Belief-State Search Tree method has no single-fault constraint, so the control node is removed and the priori probabilities are moved into the fault nodes. For every observable component an observable node is added, connected to the corresponding fault.

Since Mossberg’s model was created with a single-fault assumption, the CPTs for the nodes in the lower level only contain the probability that an observation is made given that one certain component is faulty. With the single-fault assumption relaxed a probability is needed for every combination of faulty components, i.e. every combination of faults.

To get a functional probabilistic model, probabilities of making an observation given a combination of faults are extrapolated from the probability of making an observation given a certain fault. This has been done by setting the probability for an observation, given a combination of faults, to the maximal value of having the observation, given any of the faults in the combination. For
example, if $P(O|A) = a$ and $P(O|B) = b$, we set $P(O|A, B) = \max(a, b)$. In other words, we assume that two faults or faulty components do not lead significantly more to a certain observation, than only one fault would have done.

Let $O$ be the event that an observation is made and let $Q_i$ be the component that has the fault $F_i$. Then the probability of $O$ given the combination of faults $\varphi$ is

$$P(O|\varphi) = \max_{F_i \in \varphi, F_i \text{active}} (P(O|\text{no faulty components}), P(O|Q_i = \text{faulty})) \quad (7.1)$$

By using this method to extrapolate probabilities, the probability for an observation given single faults remain the same, and in the case of multiple faults the most dominating fault decides the probability.

The repair actions in Mossberg’s model correspond to fault-revoking actions. We do not distinguish between testing actions and observation actions, but refer to both kinds as testing actions.
Chapter 8
Evaluation

In this chapter the methods developed in this master thesis are evaluated and compared to some existing methods. First, different implementations of our own methods are compared. Then the best of these implementations is compared to the implementation of the LMS method and the Dezide Advisor tool.

8.1 Evaluation of the Algorithm

8.1.1 The Test Case

Normally an initial belief-state is calculated from the initial observations, but for the XPI-model many of the possible initial observations lead to very similar belief-states. To have a larger variety of problems the step of reading the initial observations is bypassed and the initial belief-state is generated randomly. Since the objective here is to evaluate the algorithm it is of value to have a great variation of problems. To control the complexity of the problem a certain number of faults are chosen randomly. The initial belief-states are not always belief-states that are likely to appear in real test cases but when using these it allows us to evaluate the performance of the algorithm on a wider range of problems, since the objective in this experiment is evaluate the algorithm and not the model.

8.1.2 Full Search vs. Belief-State Search Tree

In the first experiment we wish to compare the Full Search method (FS) with the Belief-State Search Tree method (BSST). We also wish to evaluate the effect of using different sorting methods in the BSST method. A more accurate sorting method saves time since less nodes are expanded, but costs time when calculating what node to expand first.

The FS method is not implemented exactly as described in Chapter 4, instead it is replaced by a version of the BSST that does not prune anything. This solution is chosen simply because of not having to write a different simulation program for this method. It is a sufficiently accurate approximation since the FS method and a non pruned BSST have the same time complexities.
The BSST method is implemented in three versions of the algorithm R-BBDFS with three different sorting methods, $h_1$, $h_2$ and randomized sort, R-BBDFS-$h_1$, R-BBDFS-$h_2$ and R-BBDFS-random respectively.

**System with One Fault**

In the first experiment the two methods are tested on a model with only one fault and its associated observables, and a random initial belief-state. That is, there is a possibility $p$ that the fault is active, but since the system with the probability $1 - p$ is fault free, simply repairing the fault is not necessarily the optimal solution. It may be optimal to first perform one or several tests, to see if the device is faulty or not.

The fault is chosen randomly from the 17 faults in the XPI-model. The results presented are the mean values of the results from 25 different simulations with randomly chosen faults and randomly chosen initial belief-states. The measured values are the execution time in seconds (time), the number of expanded nodes (nodes) and, for the three R-BBDFS algorithms, the branch from the root node in which the optimal solution is found (branch). Each action going out from the initial belief state forms its own branch, and the branches are numbered in the order that the nodes are expanded. The value branch is used to measure the performance of the sorting criterion.

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>nodes</th>
<th>branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>0.148</td>
<td>77.2</td>
<td>-</td>
</tr>
<tr>
<td>R-BBDFS-$h_1$</td>
<td>0.0703</td>
<td>21.1</td>
<td>1.80</td>
</tr>
<tr>
<td>R-BBDFS-$h_2$</td>
<td>0.0784</td>
<td>15.1</td>
<td>1.16</td>
</tr>
<tr>
<td>R-BBDFS-random</td>
<td>0.0663</td>
<td>19.7</td>
<td>1.64</td>
</tr>
</tbody>
</table>

For this very simple problem all algorithms performed reasonably well, but of course FS expanded significantly more nodes and therefore had a longer execution time.

**System with Two Faults**

This experiment is performed on systems with two possible faults, and the algorithms are tested on ten different initial belief-states. In all other respects it is identical with the previous experiment.

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>nodes</th>
<th>branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>631</td>
<td>$3.21 \cdot 10^5$</td>
<td>-</td>
</tr>
<tr>
<td>R-BBDFS-$h_1$</td>
<td>27.4</td>
<td>$6.60 \cdot 10^3$</td>
<td>3.1</td>
</tr>
<tr>
<td>R-BBDFS-$h_2$</td>
<td>18.5</td>
<td>$3.94 \cdot 10^3$</td>
<td>1.2</td>
</tr>
<tr>
<td>R-BBDFS-random</td>
<td>31.0</td>
<td>$7.56 \cdot 10^3$</td>
<td>2.8</td>
</tr>
</tbody>
</table>

Now it is clear that the pruning algorithms perform significantly better, only a fraction of the full search tree is searched.

**Three Possible Faults**

In the third experiment, now with systems containing three faults, only the algorithms implementing the BSST method are tested, since the problems are too complex to be solved with a full search.
All three algorithms expand about the same number of nodes and have about the same execution time, but the complexity level of these problems are on the edge of becoming too complex to be solved. One noticeable result is that R-BBDFS-h2 always finds the optimal solution in one of the first two branches searched. This suggests that this heuristic could be used to obtain an effective estimate of the minimal expected cost of repair. If the branching factor would be limited the solution found would have a great chance of being the real optimum.

### 8.1.3 Limiting the Branching Factor and the Search Depth

For the experiments in this section the effects of limiting the branching factor and search depth are studied. The *branch limit* indicates the maximum number of children of an OR-node that are allowed to be searched. The result of experiments with two and three possible faults with different limits are presented in Tables 8.1 and 8.2. All values presented are averages for 10 different problems. The column *error* is the difference in percent between the resulting minimum expected cost $C_{min}$ and the real optimum $C^*_{min}$. It is calculated as

$$error = 100 \cdot \frac{C_{min} - C^*_{min}}{C^*_{min}}$$

The column *correct action* indicates the percentage of the times when the algorithm recommends the correct action, i.e. the first action of the optimal tree.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>time</th>
<th>nodes</th>
<th>branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-BBDFS-h₁</td>
<td>1702</td>
<td>$3.43 \cdot 10^5$</td>
<td>3.6</td>
</tr>
<tr>
<td>R-BBDFS-h₂</td>
<td>1638</td>
<td>$3.19 \cdot 10^5$</td>
<td>1.8</td>
</tr>
<tr>
<td>R-BBDFS-random</td>
<td>1862</td>
<td>$4.04 \cdot 10^5$</td>
<td>4.2</td>
</tr>
</tbody>
</table>

1The optimal value, as obtained in the previous simulations.
Table 8.1: Experiment with branch limit and search depth limit on problems with two possible faults.

<table>
<thead>
<tr>
<th>sort criterion</th>
<th>branch limit</th>
<th>depth limit</th>
<th>error</th>
<th>correct action</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_2$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>0%</td>
<td>100%</td>
<td>18.5</td>
<td>3940</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>$\infty$</td>
<td>0.11%</td>
<td>40%</td>
<td>8.20</td>
<td>2160</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>$\infty$</td>
<td>0%</td>
<td>100%</td>
<td>1.71</td>
<td>382.0</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>$\infty$</td>
<td>0.83%</td>
<td>40%</td>
<td>3.26</td>
<td>826</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>$\infty$</td>
<td>0.024%</td>
<td>100%</td>
<td>0.42</td>
<td>88.8</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>8</td>
<td>2.48%</td>
<td>40%</td>
<td>0.49</td>
<td>116</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>8</td>
<td>0.33%</td>
<td>100%</td>
<td>0.12</td>
<td>19.1</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>8</td>
<td>0.83%</td>
<td>40%</td>
<td>3.07</td>
<td>809</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>8</td>
<td>0.024%</td>
<td>100%</td>
<td>0.43</td>
<td>88.8</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>8</td>
<td>2.48%</td>
<td>40%</td>
<td>0.48</td>
<td>115</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>8</td>
<td>0.33%</td>
<td>100%</td>
<td>0.12</td>
<td>19.1</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>6</td>
<td>0.11%</td>
<td>100%</td>
<td>5.72</td>
<td>1450</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>6</td>
<td>0%</td>
<td>100%</td>
<td>1.50</td>
<td>340</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>6</td>
<td>0.83%</td>
<td>40%</td>
<td>2.16</td>
<td>554</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>6</td>
<td>0.024%</td>
<td>100%</td>
<td>0.40</td>
<td>86.4</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>6</td>
<td>3.46%</td>
<td>40%</td>
<td>0.46</td>
<td>114</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>6</td>
<td>0.33%</td>
<td>100%</td>
<td>0.12</td>
<td>19.0</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>4</td>
<td>0.11%</td>
<td>40%</td>
<td>1.69</td>
<td>406</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>4</td>
<td>0%</td>
<td>100%</td>
<td>0.77</td>
<td>162</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>4</td>
<td>0.83%</td>
<td>40%</td>
<td>0.83</td>
<td>191</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>4</td>
<td>0.024%</td>
<td>100%</td>
<td>0.30</td>
<td>58.0</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>4</td>
<td>4.25%</td>
<td>40%</td>
<td>0.29</td>
<td>60.9</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>4</td>
<td>1.11%</td>
<td>100%</td>
<td>0.11</td>
<td>16.0</td>
</tr>
</tbody>
</table>
Table 8.2: Experiment with branch limit and search depth limit on problems with three possible faults.

<table>
<thead>
<tr>
<th>sort criterion</th>
<th>branch limit</th>
<th>depth limit</th>
<th>error</th>
<th>correct action</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_2$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>0%</td>
<td>100%</td>
<td>1640</td>
<td>319000</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>$\infty$</td>
<td>0.19%</td>
<td>70%</td>
<td>866</td>
<td>176000</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>$\infty$</td>
<td>0%</td>
<td>100%</td>
<td>122</td>
<td>25000</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>$\infty$</td>
<td>0.96%</td>
<td>40%</td>
<td>98.1</td>
<td>209000</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>$\infty$</td>
<td>0.08%</td>
<td>90%</td>
<td>10.5</td>
<td>2140</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>$\infty$</td>
<td>3.06%</td>
<td>20%</td>
<td>4.33</td>
<td>80</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>$\infty$</td>
<td>1.51%</td>
<td>90%</td>
<td>0.82</td>
<td>132</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>8</td>
<td>0.13%</td>
<td>80%</td>
<td>301</td>
<td>56100</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>8</td>
<td>0%</td>
<td>100%</td>
<td>72.1</td>
<td>13800</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>8</td>
<td>0.96%</td>
<td>40%</td>
<td>51.3</td>
<td>10200</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>8</td>
<td>0.08%</td>
<td>90%</td>
<td>8.75</td>
<td>1690</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>8</td>
<td>2.95%</td>
<td>20%</td>
<td>3.12</td>
<td>560</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>8</td>
<td>1.41%</td>
<td>90%</td>
<td>0.74</td>
<td>114</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>6</td>
<td>0.14%</td>
<td>80%</td>
<td>76.6</td>
<td>12500</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>6</td>
<td>0.011%</td>
<td>100%</td>
<td>25.0</td>
<td>4090</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>6</td>
<td>1.12%</td>
<td>40%</td>
<td>17.5</td>
<td>2860</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>6</td>
<td>0.09%</td>
<td>100%</td>
<td>4.40</td>
<td>715</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>6</td>
<td>5.78%</td>
<td>30%</td>
<td>2.12</td>
<td>324</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>6</td>
<td>1.32%</td>
<td>90%</td>
<td>0.58</td>
<td>70.2</td>
</tr>
<tr>
<td>$h_1$</td>
<td>4</td>
<td>4</td>
<td>5.11%</td>
<td>60%</td>
<td>9.75</td>
<td>1380</td>
</tr>
<tr>
<td>$h_2$</td>
<td>4</td>
<td>4</td>
<td>4.69%</td>
<td>90%</td>
<td>4.83</td>
<td>681</td>
</tr>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>4</td>
<td>9.14%</td>
<td>40%</td>
<td>3.63</td>
<td>495</td>
</tr>
<tr>
<td>$h_2$</td>
<td>3</td>
<td>4</td>
<td>5.62%</td>
<td>90%</td>
<td>1.51</td>
<td>190</td>
</tr>
<tr>
<td>$h_1$</td>
<td>2</td>
<td>4</td>
<td>16.8%</td>
<td>20%</td>
<td>0.86</td>
<td>94.8</td>
</tr>
<tr>
<td>$h_2$</td>
<td>2</td>
<td>4</td>
<td>7.74%</td>
<td>90%</td>
<td>0.43</td>
<td>39.4</td>
</tr>
</tbody>
</table>

The algorithm with the sorting criterion $h_2$ clearly performs better in all cases, especially as the complexity of the problems rises. Notice that a tight branch limit for R-BBDFS-$h_2$ has little effect on the error and recommended actions but it reduces the execution time and the number of expanded nodes significantly.

For the more complex problems the depth limit has a great impact on the error. This is understandable since the repair policies need to be longer in order to guarantee that the system is repaired, when there are many possible faults. Even though the error in the estimated expected cost of repair is high the recommended actions are often correct.

These last two experiments show that a good sorting method is important for the performance of the algorithm, and that limiting the branching factor is more effective compared to limiting the search depth in reducing the time complexity with little increase of the error.
8.1.4 Iterative Deepening

To gain control over execution time the method of iterative deepening is proposed. For this experiment the algorithm R-BBDFS-H$_2$ is tested with and without iterative deepening with different branch limits on 70 problems with rising complexity. The time limit is set to 5 minutes and the error in expected cost for the different cases is compared. Since the problem of finding an exact optimum becomes hard for problems with many faults, the optimum is estimated with results from a iterative deepening run where the time limit is set to 2 hours and the branch limit is set to 4.

The iterative deepening algorithm performs generally slightly better than the one that has no depth limit. When the branch limit is high the error is very low for problems with low complexity but the error increases much as the complexity rises. This is due to that the algorithms do not have enough time to finish with the fairly strict time limit of 5 minutes. When the branch limit is restricted to 2 the error is evenly high when the complexity of the problems rises.
Figure 8.1: The average error in the estimated expected cost of repair is shown as a function of the number of possible faults in the initial belief state. The solid line is R-BBDFS-H$_2$ with iterative deepening that is allowed to run for 5 minutes on each problem. The dashed line is the same algorithm without depth-limit.
8.2 Comparison with the LMS Method and Dezide Advisor

In the master thesis of Sundberg [18] five experiments are made to evaluate the LMS method on the XPI-model. In these experiments the complete repair process is simulated. For each experiment the system has a predefined fault not known to the trouble-shooting algorithm. Predefined initial observations are given in the form of DTCs. In the simulation the recommended actions are performed until the system is fully repaired.

These five experiments are run under the same conditions with the Dezide Advisor software and the Belief-State Search Tree method. The BSST method is run using an iterative deepening version of R-BBDFS with branch limit 3.

We have chosen to use these five experiments, to get a broad base for comparing our method with other ones. A criticism to this decision is that even if many simulations are run on each test case, five test cases are not very many. Using a low number of test cases, it is not guaranteed that the tests give a just judgement of the tested methods. A larger number of test cases, with randomly chosen faults, and randomly chosen starting conditions (in this case the starting conditions are DTCs), would give a clearer and more just comparison.

8.2.1 Experiment A

In experiment A component 15 is set to be faulty and the initial observations are that DTC 3 is active and all other DTCs are inactive. The initial probability for each fault using the single fault assumption is

\[
\text{Fault:} \quad 10 \quad 12 \quad 13 \quad 15 \quad 16 \\
\text{Probability:} \quad 18.1\% \quad 20.5\% \quad 20.5\% \quad 20.5\% \quad 20.5\%
\]

and the mean cost of repair for the three different methods are

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Cost of Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSST</td>
<td>910</td>
</tr>
<tr>
<td>Dezide</td>
<td>1328</td>
</tr>
<tr>
<td>LMS</td>
<td>1625</td>
</tr>
</tbody>
</table>

It is possible to see that the BSST solved the problem with less cost. The repair sequences for all three methods are shown in Table 8.3. Testing actions \( t_i \) with an index up to 7 are actions that are equivalent to observing a component in the methods based on Heckerman’s method. Testing actions \( t_8 \)–\( t_{20} \) are equivalent to performing a test and \( t_{21} \) is the action observing the system functionality. The BSST method tends to observe components before performing the less reliable tests and waits with performing the expensive test \( t_{21} \). In the LMS method the less reliable tests are performed early since they have to be executed immediately if they are of any value. The main reason that the other methods have a higher cost is that they are forced to perform \( t_{21} \) every time a fault revoking action is performed.
Belief-State Search Tree

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>776</td>
<td>72%</td>
<td>$t_6:F^{-}t_5:F^{-}t_{17}:T^{-}r_{15}^{-}t_{11}:F^{-}t_{21}:T^{-}$</td>
</tr>
<tr>
<td>A2</td>
<td>1120</td>
<td>8%</td>
<td>$t_6:F^{-}t_5:F^{-}t_{17}:T^{-}r_{15}^{-}t_{11}:T^{-}r_{16}^{-}t_{12}:F^{-}t_{21}:T^{-}$</td>
</tr>
<tr>
<td>A3</td>
<td>1312</td>
<td>20%</td>
<td>$t_6:F^{-}t_5:F^{-}t_{17}:F^{-}r_{16}^{-}t_{11}:T^{-}t_{16}:T^{-}r_{7}:T^{-}r_{15}^{-}t_{21}:T^{-}$</td>
</tr>
</tbody>
</table>

Dezide Advisor

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1328</td>
<td>100%</td>
<td>$r_{16}^{-}t_{21}:F^{-}t_6:F^{-}t_7:T^{-}r_{15}^{-}t_{21}:T^{-}$</td>
</tr>
</tbody>
</table>

LMS

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>968</td>
<td>42%</td>
<td>$t_{16}:T^{-}t_5:F^{-}t_6:F^{-}t_7:T^{-}r_{15}^{-}t_{21}:T^{-}$</td>
</tr>
<tr>
<td>A2</td>
<td>985</td>
<td>20%</td>
<td>$t_{16}:F^{-}t_{17}:T^{-}t_5:F^{-}t_6:F^{-}t_7:T^{-}r_{15}^{-}t_{21}:T^{-}$</td>
</tr>
<tr>
<td>A3</td>
<td>2688</td>
<td>38%</td>
<td>$t_{16}:F^{-}t_{17}:F^{-}r_{16}^{-}t_{21}:F^{-}t_5:F^{-}t_6:F^{-}r_{10}^{-}t_{21}:F^{-}t_7:T^{-}r_{15}^{-}t_{21}:T^{-}$</td>
</tr>
</tbody>
</table>

Table 8.3: Repair sequences for experiment A.

8.2.2 Experiment B

In this experiment component 16 is set to be faulty and the DTCs 1 and 3 are active. The initial probability for each fault using the single fault assumption is

<table>
<thead>
<tr>
<th>Fault</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>18.1%</td>
</tr>
<tr>
<td>12</td>
<td>20.5%</td>
</tr>
<tr>
<td>13</td>
<td>20.5%</td>
</tr>
<tr>
<td>15</td>
<td>20.5%</td>
</tr>
<tr>
<td>16</td>
<td>20.5%</td>
</tr>
</tbody>
</table>

The mean cost of repair for the three different methods are

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Cost of Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSST</td>
<td>669</td>
</tr>
<tr>
<td>Dezide</td>
<td>410</td>
</tr>
<tr>
<td>LMS</td>
<td>670</td>
</tr>
</tbody>
</table>

The repair sequences for this experiment are shown in Table 8.4. In this experiment Dezide Advisor has the lowest cost. This is because the efficiency of repairing component 16 is very high. BSST waits with repairing this fault until it is more sure that this is the actual fault in order to keep the expected cost minimal.
8.2.3 Experiment C

In this experiment component 17 is set to be faulty and DTC 1 is the only active DTC. The initial probability for each fault using the single fault assumption is 2

<table>
<thead>
<tr>
<th>Fault:</th>
<th>Probability:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>17.3%</td>
</tr>
<tr>
<td>3</td>
<td>17.3%</td>
</tr>
<tr>
<td>4</td>
<td>17.3%</td>
</tr>
<tr>
<td>5</td>
<td>20.4%</td>
</tr>
<tr>
<td>6</td>
<td>8.6%</td>
</tr>
<tr>
<td>7</td>
<td>7.1%</td>
</tr>
<tr>
<td>8</td>
<td>8.6%</td>
</tr>
<tr>
<td>17</td>
<td>3.5%</td>
</tr>
</tbody>
</table>

and the mean cost of repair for the three different methods are

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Cost of Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSST</td>
<td>1872</td>
</tr>
<tr>
<td>Dezide</td>
<td>3091</td>
</tr>
<tr>
<td>LMS</td>
<td>4627</td>
</tr>
</tbody>
</table>

The repair sequences for this experiment are shown in Table 8.5. The resulting repair sequences are not presented in Sundberg’s master thesis for this and the following experiments. Here BSST performs significantly better. This is due to that the fault of the system has a very low probability given the initial observations and that many of the faults with higher probability are not directly observable. Dezide and LMS must perform the expensive testing action $t_{21}$ several times while BSST waits with this test until the very end since repairing several faults in a row is optimal.

2Faults with a probability less than 2% are ignored.
### Belief-State Search Tree

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>1869</td>
<td>96%</td>
<td>r₄-r₃-r₄-r₇-r₈-r₂-t₂₁:T-t₂₁:F-t₁:F-t₉:T-t₁₇:T</td>
</tr>
<tr>
<td>C₂</td>
<td>1952</td>
<td>4%</td>
<td>r₄-r₃-r₄-r₇-r₈-r₂-t₂₁:T-t₂₁:F-t₁:F-t₉:F-t₁₁:F-t₁₀:F-t₁₇-t₂₁:T</td>
</tr>
</tbody>
</table>

### Dezide Advisor

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>2885</td>
<td>84%</td>
<td>r₂-t₂₁:F-r₄-t₂₁:F-t₁₇:F-r₃-t₂₁:F-r₈-t₂₁:F-t₁₁:F-r₃-t₂₁:F-r₈-t₂₁:F-t₁₇-t₂₁:T</td>
</tr>
<tr>
<td>C₂</td>
<td>3503</td>
<td>5.5%</td>
<td>r₂-t₂₁:F-r₄-t₂₁:F-t₁₇:T-t₁:F-t₂:F-r₆-t₂₁:F-t₁₁:F-r₃-t₂₁:F-r₈-t₂₁:F-t₁₇-t₂₁:T</td>
</tr>
<tr>
<td>C₃</td>
<td>4523</td>
<td>10.5%</td>
<td>r₂-t₂₁:F-r₄-t₂₁:F-t₁₇:F-r₃-t₂₁:F-r₈-t₂₁:F-r₇-t₂₁:F-t₁₁:F-t₁₄-t₂₁:F-r₁₆-t₂₁:F-t₉:T-r₁₇-t₂₁:T</td>
</tr>
</tbody>
</table>

### LMS

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.5: Repair sequences for experiment C.

### 8.2.4 Experiment D

In this experiment component 4 is set to be faulty and DTC 1 is the only active DTC. The initial probability for each fault using the single fault assumption is

<table>
<thead>
<tr>
<th>Fault</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>17.3%</td>
</tr>
<tr>
<td>3</td>
<td>17.3%</td>
</tr>
<tr>
<td>4</td>
<td>17.3%</td>
</tr>
<tr>
<td>5</td>
<td>20.4%</td>
</tr>
<tr>
<td>6</td>
<td>8.6%</td>
</tr>
<tr>
<td>7</td>
<td>7.1%</td>
</tr>
<tr>
<td>8</td>
<td>8.6%</td>
</tr>
<tr>
<td>17</td>
<td>3.5%</td>
</tr>
</tbody>
</table>

and the mean cost of repair for the three different methods are

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Cost of Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSST</td>
<td>674</td>
</tr>
<tr>
<td>Dezide</td>
<td>581</td>
</tr>
<tr>
<td>LMS</td>
<td>967</td>
</tr>
</tbody>
</table>

The repair sequences for this experiment are shown in Table 8.6. Here Dezide performs better because of the mandatory system functionality observation, the t₂₁ test. BSST continues to repair faults after the actual fault already has been repaired since according to the BSST method it is better for the total expected cost to wait with observing the system functionality.
Belief-State Search Tree

<table>
<thead>
<tr>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
</table>

Dezide Advisor

<table>
<thead>
<tr>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>581</td>
<td>r_2–t_21:F–r_3–t_21:T</td>
</tr>
</tbody>
</table>

LMS

<table>
<thead>
<tr>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.6: Repair sequences for experiment D.

8.2.5 Experiment E

In this experiment component 9 is set to be faulty and DTC 8 is the only active DTC. The initial probability for each fault using the single fault assumption is

<table>
<thead>
<tr>
<th>Fault</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>7.5%</td>
</tr>
<tr>
<td>11</td>
<td>92.5%</td>
</tr>
</tbody>
</table>

and the mean cost of repair for the three different methods are

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Cost of Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSST</td>
<td>501</td>
</tr>
<tr>
<td>Dezide</td>
<td>870</td>
</tr>
<tr>
<td>LMS</td>
<td>693</td>
</tr>
</tbody>
</table>

The repair sequences for this experiment are shown in Table 8.7. There are only two possible faults, of which the actual fault is observable. The testing action \( t_4 \) that observes component 9 costs very little, but requires a transition in the decomposition state. Repairing component 11 costs more but it requires the same decomposition state. Since the same decomposition state is needed, the BSST-method decides it might as well perform the cheap test \( t_4 \) to rule out the possibility that component 9 is faulty. In this case making this test proves to be a successful decision. If the fault had been in component 11 the extra cost of performing this test would have been marginal.

Belief-State Search Tree

<table>
<thead>
<tr>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>501</td>
<td>t_4:T–r_9–t_21:T</td>
</tr>
</tbody>
</table>

Dezide Advisor

<table>
<thead>
<tr>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>870</td>
<td>r_{11}–t_{21}:F–r_9–t_{21}:T</td>
</tr>
</tbody>
</table>

LMS

<table>
<thead>
<tr>
<th>Cost</th>
<th>Prob.</th>
<th>Repair Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.7: Repair sequences for experiment E.

Concluding, the limited number of comparing tests prevent us from drawing any definite conclusions of which method is the best. However, these test results suggest that the BSST method might be the best choice, if the number of
possible faults are many, as in experiment C. We have been able to see that using the system functionality observation can lead to finding the fault earlier in the repair process, but can also lead to high costs if the fault is not found.
Chapter 9

Conclusions and Results

As explained in Chapter 6 the size of the state space makes the problem intractable if it would be solved as a POMDP. Because of the small support the size of the state space can be reduced significantly after the initial observations and since the number of reachable belief-states is limited the problem can be solved by searching the belief-state space.

Searching the belief-state space with the R-BBDFS algorithm on a AND/OR tree made up of belief-states, actions and observations as proposed in Chapter 5 guarantees that the optimal solution will be found on the condition that there is no limit to the branching factor or the search depth. However, the complexity of this algorithm grows exponentially with the size of the problem.

The evaluation of the algorithm in Section 8.1 on the model of the XPI-system shows that when the nodes that are to be searched are sorted according to a heuristic that gives a more accurate approximation of the solution the algorithm performs faster, even though calculating the approximation comes with a larger computational overhead.

When entropy in combination with the lower bound is used as the heuristic that approximates the cost of the optimal solution rather than just using the lower bound the optimal solution is often found among the first couple of branches that are searched. This means that it is possible to limit the branching factor with a small error. This limit leads to a great reduction of the execution time of the algorithm.

When the depth is limited the search algorithm performs in polynomial time instead of exponential time. In the experiments where the search depth is limited the resulting solution has an expected cost that is much too high when the optimal policy tree has a depth greater than this limit. However, the same experiments show that the recommended action often is the optimal action to perform. When iterative deepening is used instead of a fixed depth limit we gain more control of the execution time of the algorithm. This is useful when it is important to receive a solution quickly.

In Section 8.2 the BSST method is compared to the LMS method and the commercial troubleshooting tool Dezide Advisor. The experiments used in the comparison were originally designed to evaluate the LMS method on the XPI-model in another master thesis written on Scania [18]. The BSST often performed better than the other two when troubleshooting the XPI-model. The reason is that methods based on Heckerman’s method only are optimal when
the assumptions stated in Section 6.1 are true.

When troubleshooting a complex system like a truck, the method developed in this master thesis compares well to other existing methods today.
Chapter 10

Future Improvements

From one answer many questions can arise. As we have investigated the problem of troubleshooting a complex system such as a truck there is so much more we would want to test and investigate.

It would be interesting to see how this algorithm performs in comparison with the other methods on cases where there are more than one fault. On the large systems the effect of using the entropy based heuristic can be investigated. Perhaps it is possible to create a hybrid between this method and Heckerman’s method.

The algorithm itself can be improved with smarter programming. As for now a belief-state with 10 possible faults is represented with a vector containing $2^{10}$ elements, but for fault-combinations with more than 3 faults the probability is practically zero. If the belief-state would be approximated to contain only fault combinations with 3 or less active faults the algorithm would perform much faster. The AND/OR tree that is searched can perhaps be replaced with an AND/OR graph and maybe it is possible to use a version of the A* algorithm adapted for the search of AND/OR trees.

In order to know how exact the probabilistic model needs to be it would be interesting to see how sensitive the different troubleshooting methods are to errors in the model.
Bibliography


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Håkan Warnquist, Petter Säby