Linköping Studies in Science and Technology

Dissertations. No. 1691

# Troubleshooting Trucks – Automated Planning and Diagnosis

by

**Håkan Warnquist**

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2015

ISBN 978-91-7685-993-3
ISSN 0345–7524
Printed by LiU Tryck 2015

URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-119445

Cover image courtesy of Scania CV AB (publ)

# Abstract

This thesis considers computer-assisted troubleshooting of heavy vehicles such as trucks and buses. In this setting, the person that is troubleshooting a vehicle problem is assisted by a computer that is capable of listing possible faults that can explain the problem and gives recommendations of which actions to take in order to solve the problem such that the expected cost of restoring the vehicle is low. To achieve this, such a system must be capable of solving two problems: the diagnosis problem of finding which the possible faults are and the decision problem of deciding which action should be taken.

The diagnosis problem has been approached using Bayesian network models. Frameworks have been developed for the case when the vehicle is in the workshop only and for remote diagnosis when the vehicle is monitored during longer periods of time.

The decision problem has been solved by creating planners that select actions such that the expected cost of repairing the vehicle is minimized. New methods, algorithms, and models have been developed for improving the performance of the planner.

The theory developed has been evaluated on models of an auxiliary braking system, a fuel injection system, and an engine temperature control and monitoring system.

*This work has been supported Scania CV AB and FFI – Fordonsstrategisk Forskning och Innovation.*

# Acknowledgments

Denna avhandling är slutet på en lång och spännande resa som inte hade varit möjligt utan hjälp. Arbetet med avhandlingen har varit en del av ett forskningssamarbete mellan fordonstillverkaren Scania och institutionen för datavetenskap vid Linköpings universitet.

I Linköping vill jag tacka mina handledare Professor Patrick Doherty och Dr Jonas Kvarnström. Patrick har gett mig god träning i och verktyg för hur jag muntligt presenterar min forskning. Jonas har varit ett otroligt stöd vid framtagandet av nya forskningsidéer och forskningsartiklar. Med nästan övermänsklig noggrannhet kan hans skarpa ögon upptäcka minsta oklarhet i snåriga ekvationer. Jag är tacksam för all kunskap han delat med sig av. Jag har lärt mig mycket om hur man skriver vetenskapliga artiklar. Stort tack till Anne Moe, koordinator för doktorander på institutionen för datavetenskap. Hon är en klippa och har gett mig ovärderlig hjälp i den praktiska planeringen av framläggningarna av denna avhandling. Jag vill även tacka alla doktorander på KPLAB för alla spännande lunchdiskussioner.

På Scania vill jag först och främst tacka Dr Mattias Nyberg som varit initiativtagare till forskningsprojekten och som gett mig förstklassig handledning. Mattias har tillsammans med Dr Jonas Biteus varit med och utvecklat tillämpningen av min forskning på Scania. Jag vill även tacka alla andra idésprutor som bidragit, i synnerhet Hans Ivendal, Dr Tony Lindgren, Dr Martin Eineborg och övriga i min arbetsgrupp på Scania. Jag vill även tacka min styrgrupp och alla de skickliga chefer stöttat mig under denna tid: Göran Johansson, Thor Brenden, Per Karlsson, Anna Pernestål, Björn Bertmar och Erik Malmkvist.

Dessutom vill jag tacka företaget Scania och forskningsprogrammen Vehicle Information and Communication Technology (V-ICT) och Fordonsstrategisk Forskning och Innovation (FFI) inom Vinnova. Utan dessa finansiärer hade inte min forskning kunnat bli möjlig.

Sist men kanske mest vill jag tacka min fru, mina barn, syskon, föräldrar, svärföräldrar, nära och kära för er tveklösa kärlek. Ni har outtröttligt följt mitt arbete och på era olika vis stöttat min resa. Mina barn Aron och Klas har gett mig enorm energi och är genuint stolta över att jag arbetar med lastbilar. Min fru Sara har satt sig in i mitt arbete, kontrolläst material och lyssnat när jag övat på muntliga framställningar. Hon är fantastisk och jag älskar henne väldigt mycket.

# Populärvetenskaplig sammanfattning

Denna avhandling behandlar datorstödd felsökning av tunga fordon så som lastbilar och bussar. Moderna lastbilar och bussar består av flera mekaniska, hydrauliska, och elektroniska system som samverkar för att ge fordonet önskat beteende. När ett fel uppstår som gör att någon av fordonets högnivåfunktioner inte fungerar som den ska kan det vara svårt för en mekaniker att felsöka fordonet utan datorstöd. Ett exempel på ett sådant fel kan vara att fordonet inte uppnår fullt arbetstryck i bränsleinsprutningssystemet. Detta kan ha många orsaker, t.ex. fel på en bränsleledning, insprutare, reglerventil, bränsletryckgivare eller bränslepump. För att lösa detta måste mekanikern utföra flera mätningar och eventuellt provbyta vissa potentiellt trasiga komponenter. Detta kan bli både tidskrävande och dyrt. Mekanikern kan då behöva datorstöd för att effektivisera felsökningen.

Idag används redan datorer vid felsökning i verkstad. Dessa används framförallt för att hämta ut information ur fordonets styrenheter så som mätvärden och eventuella fellarm som utlösts av fordonets interna diagnossystem. Vid den typ av felsökning som avses i denna avhandling, använder personen som felsöker fordonet datorn för att få förklaringar på möjliga fel som kan ha orsakat de upplevda problemen och för att få stöd i vilka felsökande åtgärder som är lämpliga att ta. Datorn tolkar då samband mellan mätdata, observerade symptom och fel för att beräkna vilka av dessa fel som troligast förklarar de observationer som gjorts på just det fordon som mekanikern har framför sig. Det är möjligt att observationerna kan förklaras av flera fel. Beslutsstöd ges genom att datorn beräknar kostnadseffektiviteten av de möjliga åtgärder som kan tas. En åtgärd kan t.ex. vara en reparation eller en mätning med syftet att skilja på möjliga fel.

Problemet med att beräkna vilka fel som är troligast benämns som diagnosproblemet. Problemet med att beräkna vilken eller vilka åtgärder som är kostnadseffektivast benämns som beslutsproblemet. I denna avhandling studeras olika metoder för att lösa dessa problem i sammanhanget felsökning av lastbilar.

För diagnosproblemet har framförallt metoder baserade på Bayesianska nätverks-modeller studerats. Dessa modeller lämpar sig väl när samband mellan fel och mätbara observationer är osäkra. Detta gäller i synnerhet mekaniska fel som kan bete sig olika beroende på exakt hur den felande komponenten gått sönder. I avhandlingen har flera metoder baserade på Bayesianska nätverk utvecklats. Både för fallet då fordonet befinner sig i en verkstad och då fordonet befinner sig på väg. Det kan vara väldigt beräkningsintensivt att beräkna sannolikheten för fel givet observationer för ett fordon som beskrivits med en Bayesiansk nätverksmodell. Genom att begränsa strukturen i nätverken kan man bygga modeller som är mindre beräkningsintensiva. Metoderna som presenteras i avhandlingen är även utformade för att underlätta arbetet med att bygga och underhålla modeller för flera fordon som kan se olika ut. Detta är viktigt för att kunna tillämpa metoderna inom industrin.

Beslutsproblemet har angripits genom att skapa automatiska planerare som kan välja ut åtgärder så att den förväntade kostnaden att reparera fordonet minimeras. Denna typ av metoder är kraftfulla när det krävs en lång serie av åtgärder för att identifiera och åtgärda felet på fordonet. En nackdel är att de är notoriskt beräkningsintensiva. Nya metoder och algoritmer har utvecklats för att förbättra befintliga planeringsmetoder i det avseendet. Algoritmer för automatisk planering använder ofta sökheuristiker för att värdera olika alternativ. Detta är funktioner som ger en skattning av den förväntade reparationskostnaden innan en fullständig plan har beräknats. Ju bättre skattningen är desto effektivare blir planeraren. I avhandlingen presenteras flera nyutvecklade sökheuristiker som är tillämpbara för automatisk planering. En annan metod som presenteras i avhandlingen låter den automatiska planeraren abstrahera bort de åtgärder som behövs för att ta isär eller sätta ihop komponenter innan en särskild mätning eller reparation kan utföras. Detta gör planeringen effektivare genom att färre åtgärder behöver beaktas.

Teorin som har utvecklats har utvärderats på modeller av flera system på en modern lastbil; ett hydrauliskt bromssystem, ett bränsleinsprutningssystem, och ett system för reglering och övervakning av motortemperaturen. Resultaten är lovande inför byggandet av framtidens felsökningsstöd.

# Contents

# Chapter 1

# Introduction

In the context of mechanical and electronic systems, *troubleshooting* is the process of finding and correcting faults on such a system [131]. The purpose of this thesis has been to study and develop methods for computer-assisted troubleshooting of motor vehicles, in particular trucks and buses. In this setting, the person performing the troubleshooting is assisted by a computer which lists possible faults and recommends actions that can be taken to proceed in the troubleshooting. To achieve this, the computer must solve two problems: the *diagnosis problem* of finding which faults are likely given the current available information, and the *planning problem* of deciding which action or actions should be taken next. The person performing the troubleshooting is typically a workshop mechanic but in other scenarios it can be a help-desk operator trying to solve a vehicle problem remotely. The troubleshooting is successful if the problem-cause is remedied, and it is better if the cost of performing the troubleshooting is lower.

## 1.1 Background and Motivation

In recent years heavy duty trucks have gone from being purely mechanical systems to being complex computerized electromechanical systems [35]. Increased requirements on safety and environmental performance have led to the development of several new subsystems that are coordinated by a distributed system of electronic control units (ECU:s) that communicate internally over an in-vehicle network, e.g. a CAN bus. For example, a modern electronic braking system can perform *brake blending* by coordinating the gear box with the exhaust, hydraulic, and conventional braking systems to achieve the requested braking torque [112]. Systems such as the Exhaust Gas Recycling (EGR) system which directs parts of the exhaust gases back into the cylinders to reduce combustion temperature [114, 140] and the Selective Catalytic Reduction (SCR) system which injects urea into the exhaust gases to chemically remove nitrous gases (NOx) [117] have been developed to meet the new higher demands on environmental performance. The system for Adaptive Cruise Control (ACC) controls engine speed and braking torque to maintain the vehicle's distance to other vehicles ahead [113]. When the high-level functionality of such a complex system fails it can be difficult to identify the root cause based on the observed symptoms, e.g. excessive wear on the conventional brakes, high NOx levels in exhaust gases, or the ACC failing to maintain correct distance. A mechanic needs to understand how the system is implemented to be able to find the problem-cause. This can be difficult because many of these systems are new and integrate the functions of many components. E.g. a failure on the retarder control valve can cause the hydraulic brake to become less efficient thus increasing the load on the conventional brakes because of incorrect brake blending. Failures for which the mechanics are unable to locate the causes are a costly problem for the industry [120]. A tool for computer-assisted troubleshooting capable of pointing out suspected faults and recommending suitable troubleshooting actions can help mechanics and other decision makers solve vehicle problems more cost-efficiently.

Computers are already a frequently used tool in the workshops today. Using software tools, the workshop mechanic can connect to the On-Board Diagnostic (OBD) system on the vehicle to read out diagnostic information from the ECU:s, set parameters, and run diagnostic tests [118, 141] (see Figure 1.1). The OBD system monitors the vehicle during operation. When a failure is detected, it is signaled by generating a Diagnostic Trouble Code (DTC) which identifies particular failure types using a standardized format [130]. Ideally, each DTC exactly points out the problem-causing component and thereby removes the need for troubleshooting. This is true for many types of electrical faults, but for mechanical faults and high-level function failures, multiple problem-causes may share a common DTC, e.g. the DTC that is generated when high NOx values are measured. When troubleshooting, the mechanic will therefore also gather information from other sources such as the driver, visual inspections, and tests. If the computer-assisted troubleshooting system should be useful, it too must be able to use information from these sources.

Figure 1.1: Software tool used to connect to OBD system.

It is important that the troubleshooting is cost-efficient. The cost of troubleshooting includes the repair cost, which comes from the time spent by the mechanic, the spare parts needed, and other resources consumed such as oils and coolant. Additionally, trucks and buses are typically used in commercial applications where the total cost of operating the vehicles directly affects the profit for the vehicle owner. Therefore, when a vehicle problem occurs, it is also important to consider the downtime of the vehicle which can be much longer than the time spent in the workshop. A troubleshooting system that can help reduce both the expected repair cost and the vehicle downtime can yield great economic savings.

For example, the repair and maintenance costs for long-haulage heavy trucks can stand for as much as 10000 € per year [44]. When an unexpected breakdown occurs, the economic loss for the owner of a long-haulage truck can be on average 1000 € each time [56]. To safeguard against unexpected losses, the vehicle owner can buy a repair and maintenance contract from the service provider for a fixed price [81, 115, 142]. Some of these contracts also compensate the vehicle owner for downtime and unexpected breakdowns [116, 143]. This gives the service provider reason to provide a service solution that is cost-efficient with regards to both repair and downtime costs as this can improve their margins on these contracts.

## 1.2 Problem Formulation

We will generalize from road vehicles and look upon the object that we troubleshoot as a *system* consisting of *components*. Some of these components can have the status *faulty* and should be repaired. We do not know with certainty

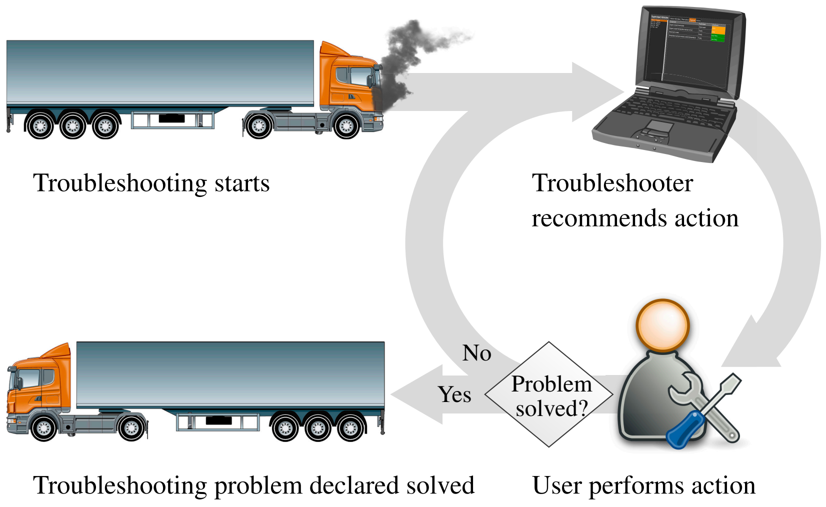
Troubleshooting starts
Troubleshooter recommends action
No
Yes
Problem solved?
Troubleshooting problem declared solved
User performs action

Figure 1.2: The troubleshooting process.

which of the components are faulty, but we can make observations from which we can draw conclusions about the status of the components. The system is said to be fault-free when none of the components which constitute the system are faulty. The *troubleshooting problem* is to sequentially select and perform one action at a time until the system is declared fault-free. The troubleshooting problem is successfully solved if the system actually is fault-free when it is declared to be fault-free.

We want a solution to the troubleshooting problem where a *user* performs actions recommended by a system for computer-assisted troubleshooting. We call this system the *troubleshooter*. Figure 1.2 shows how action recommendations by the troubleshooter are interleaved with action executions by the user until the troubleshooting problem can be declared solved. The actions performed by the user can affect the state of the system and its environment and gain information through observations. This information is used by the troubleshooter when computing the next recommendation.

When the system to troubleshoot is a truck or bus, the user can be the mechanic in the workshop or a help-desk operator performing troubleshooting remotely. The observations can consist of information such as DTC:s from the OBD system, problem descriptions by the vehicle owner, and feedback regarding which actions have been performed and their outcomes. The troubleshooter would also need product information describing the configuration of the specific vehicle at hand and operational statistics such as mileage and operational hours.

Problems formulated similarly to the troubleshooting problem are commonly separated in the literature into two parts: the *diagnostic problem* and the *decision problem* [37, 47, 62, 75, 90, 132, 146]. The diagnostic problem is about determining the status of the components given currently available information. The decision problem is about deciding which action to do next. When decisions should be planned several steps ahead, this is the *planning problem.* When solving the decision problem, one needs to understand the consequences of actions. This includes solving the diagnosis problem, and partially [62, 75] or fully [90, 146] solving the planning problem.

There are several variations of these problem formulations of which a subset is addressed in this thesis.

### 1.2.1 The Diagnostic Problem

A *diagnosis* is commonly defined as a statement that describes a reason for a problem or an act of finding diagnoses [84]. In this thesis the description of the cause of a problem will be specified in terms of a set components in the system that are faulty. The entity generating the diagnosis is the *diagnoser.* The purpose of a diagnoser can be *fault detection* or *fault isolation* [22]. For fault detection, we want to discriminate the case where no component is faulty from the case where at least one component is faulty. It is often important that faults are detected as soon as possible after a fault has occurred. For fault isolation, we want to know more specifically which diagnoses are possible. There are several possible formulations of the problem of fault isolation described in the literature.

- *Finding consistent or minimal diagnoses.* A diagnosis is said to be *consistent* if it is consistent with the model describing the system and the observations made. A consistent diagnosis is said to be *minimal* if there is no other consistent diagnosis whose set of faulty components is a subset of the set of faulty components for the minimal diagnosis [39]. When it is possible that multiple components can be faulty, the number of consistent diagnoses is exponential in the number of components. Therefore, finding all minimal diagnoses is often preferred over finding all consistent diagnoses. In particular if a model that only describes the nominal behavior of the components is used, then the set of all minimal diagnoses characterizes the set of all consistent diagnoses because any diagnosis with a larger set of faulty components is also consistent with this model [39].

- *Finding the most likely diagnosis.* Another possible goal of the fault isolation is to estimate a single diagnosis that best fits the observations. Often, a machine learning method such as Support Vector Machines [34] or Neural Networks [82] can be used (see e.g. [66, 110, 147]).

- *Finding a probability distribution for the possible diagnoses.* When a more detailed view of the possible diagnoses is needed, the purpose of fault isolation can be to compute the probability distribution over all diagnoses given the observations:
$$\Pr(\text{diagnosis}|\text{observations}). \tag{1.1}$$
When multiple faults are possible, the number of possible diagnoses grows exponentially with the number of components. Therefore the set of possible diagnoses is sometimes limited to those with at most a certain number of faulty components [78]. Another possibility is to compute only the marginal probability of each fault given the observations:
$$\Pr(\text{fault}|\text{observations}). \tag{1.2}$$

In this thesis the goal of the diagnoser is to compute probability distributions of the possible diagnoses, either as complete diagnoses (1.1) or marginalized fault probabilities (1.2). The probability distribution provides more detailed information of the health state of the vehicle which is useful when solving the decision problem. In addition to this, the diagnoser may need to compute the probability that a new observation has a particular outcome given previous observations:

$$\Pr(\text{outcome of new observation}|\text{observations}) \tag{1.3}$$

so that the consequence of making an observation can be evaluated when making decisions.

Another problem that needs to be addressed in order to build a practical and functional troubleshooter in an industrial setting is how the models needed by the troubleshooter should be created and maintained in an efficient way. The work effort needed for model creation and maintenance needs to be balanced with the accuracy and precision of the models in order to achieve a desirable performance of the troubleshooting system as a whole.

Model creation can be a manual process where models are created by domain experts and it can be an automatic process where the models are learned from data. This thesis primarily considers models that are created by domain experts, but methods for learning parameters from statistical data are also studied. A method that solely relies on model learning from statistical data risks having too little data for new vehicle types. However, learning methods can be a powerful tool to improve the quality of a manually created models over time.

Modern automotive vehicles are modular in design [91]. This means that many unique instantiations of vehicles can be created from only a set of few modules (Figure 1.3). This needs to be taken into consideration when creating the models because it is not feasible to create a unique model for every possible vehicle in-

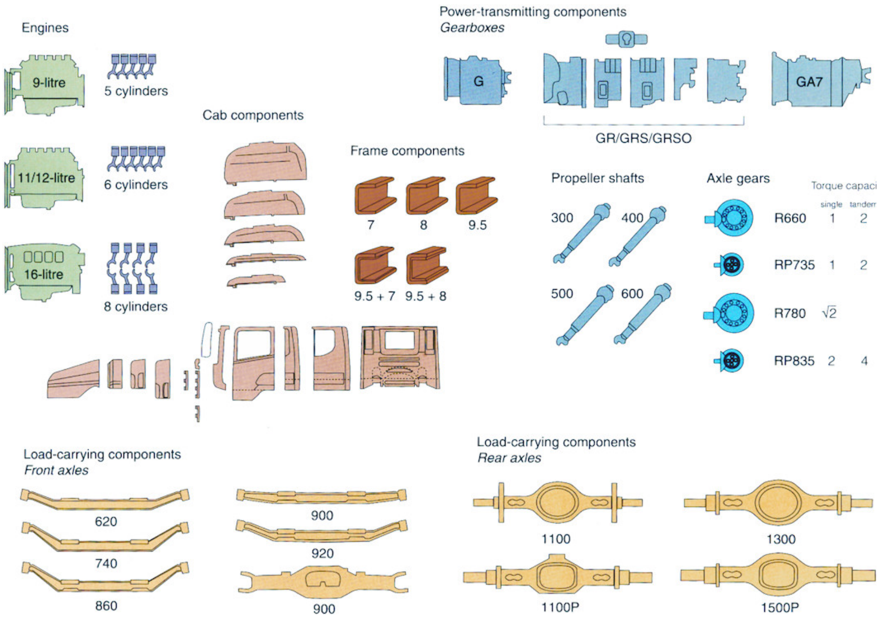

Figure 1.3: Modular design of vehicles allows many unique instantiations to be created from a set of few modules.

stantiation. Either the models are created so that they fit many instantiations of the vehicles or different models are created for each module so that it is possible to combine these into a vehicle-specific model automatically.

### 1.2.2 The Decision Problem

The actions that we can perform to solve the troubleshooting problem can belong to the following categories:

- *repair actions* that make a faulty component non-faulty for example by replacing or repairing it,
- *observing actions* that gain information by making an observation for example making a measurement, performing a test, or asking a question, and
- *other actions* that manipulate the vehicle in some other way such as preparing for a repair or an observation by disassembling obstructing parts.

It is not necessarily the case that every sequence of actions that solves the troubleshooting problem has sufficient quality to be considered good troubleshooting. For example, a sequence of actions that repairs every component on the system would certainly solve the troubleshooting problem, but it would also likely be prohibitively time-consuming and expensive. The problem of selecting the next action or sequence of actions should therefore be done with regard to a performance measure, e.g. the cost.

Each action that we can perform has a cost. We define the *cost of repair* to be the sum of the costs of all actions that are performed until the troubleshooting session can be terminated because the problem is believed to be solved. Assume that all that the troubleshooter knows of the vehicle at a given moment (what has been done and what has been observed) can be represented with a state of information $s$.

**Definition 1.1** (Cost of Repair)**.** For a given state of information $s$, the *cost of repair* is

$$CR(s) = \begin{cases} 0 & \text{if the troubleshooting session can be terminated in state } s, \\ cost(a,s) + CR(s_a) & \text{otherwise,} \end{cases} \tag{1.4}$$

where $a$ is the action performed in state $s$, $cost(a,s)$ is the cost of performing the action $a$ in state $s$, and $s_a$ is the next state after performing action $a$ in state $s$ and observing its outcome.

The state of information is affected by the outcomes of the actions and these can only be known after the action has been performed. Therefore we can only compute the cost of repair like this once we have observed the outcomes of all actions. Also, depending on the outcome of the action we can decide to perform different actions. If we know the probability distribution of the outcomes, the *expected cost of repair* (ECR) is a cost measure that can be used before-hand:

**Definition 1.2** (Expected Cost of Repair)**.** For a given state of information $s$, the *expected cost of repair* is

$$ECR(s) = \begin{cases} 0 & \text{if the troubleshooting session can be terminated in state } s, \\ cost(a,s) + \sum_{o \in \Omega_a} \Pr(o|a,s) ECR(s_{o,a}) & \text{otherwise,} \end{cases} \tag{1.5}$$

where $a$ is the action performed in state $s$, $cost(a,s)$ is the cost of performing the action $a$ in state $s$, $\Omega_a$ is the set of possible outcomes of the action $a$, $\Pr(o|a,s)$ is the probability that the action $a$ has the outcome $o$ in state $s$, and $s_{o,a}$ is the state after action $a$ is performed and outcome $o$ is seen.

In this thesis the ECR is the primary performance measure for the decision problem. In all troubleshooting it is desirable that the ECR is small and in *optimal troubleshooting* the ECR is minimal. The minimal ECR is:

$$ECR^*(s) = \begin{cases} 0 & \text{if the troubleshooting session can be terminated in state } s, \\ \min_{a \in \mathcal{A}} cost(a,s) + \sum_{o \in \Omega_a} \Pr(o|a,s) ECR^*(s_{o,a}) & \text{otherwise.} \end{cases} \tag{1.6}$$

where $\mathcal{A}$ is the set of all possible actions that can be performed.

The decision problem can either be solved *offline* or *online*. When solved offline, a full conditional plan of actions is created before the troubleshooting is started. When solved online, a conditional plan of actions is only partially created, if at all, each time the troubleshooter is asked to recommend an action. Figure 1.4 shows the full conditional plan of actions for a very small example of troubleshooting problems with the engine temperature. In general, the problem of deciding which actions to perform so that optimal troubleshooting is achieved is computationally intractable [100]. Instead, the decision problem is solved suboptimally by trading off the ECR with computation time, e.g. by only computing the plan a limited number of steps ahead and then using a heuristic to estimate the remaining ECR. In *near-optimal troubleshooting*, decisions are made so that the difference between the achieved ECR and the optimal ECR is within a provable bound. In this thesis, we typically consider the problem of near-optimal troubleshooting. When optimal solutions are computed these are only used as reference.

The difficulty of solving the decision problem also depends on how the cost and effects of actions are modeled. For example, the action cost can be modeled to be a unit cost that is the same for all actions, a constant cost specific for each action, or a cost that is dependent on the state of the system or on the order in which the actions are performed. Actions can also have constraints that limit when they can be performed.

In this thesis we consider the decision problem for two different scenarios of vehicle troubleshooting:

- *Workshop troubleshooting* where the vehicle is located at a workshop and the possible actions are primarily actions for performing repairs and observations. Sometimes we also include assembly and disassembly actions that gain access to different components and measurement points on the vehicle and an action for ending an ongoing troubleshooting session.

- *Integrated remote and workshop troubleshooting* where the vehicle can either be on the road or in the workshop and we have additional actions for running remote tests, making reservations at workshops, and transporting the vehicle to a workshop. The integrated remote and workshop troubleshooting problem differs from the workshop troubleshooting problem in that there are

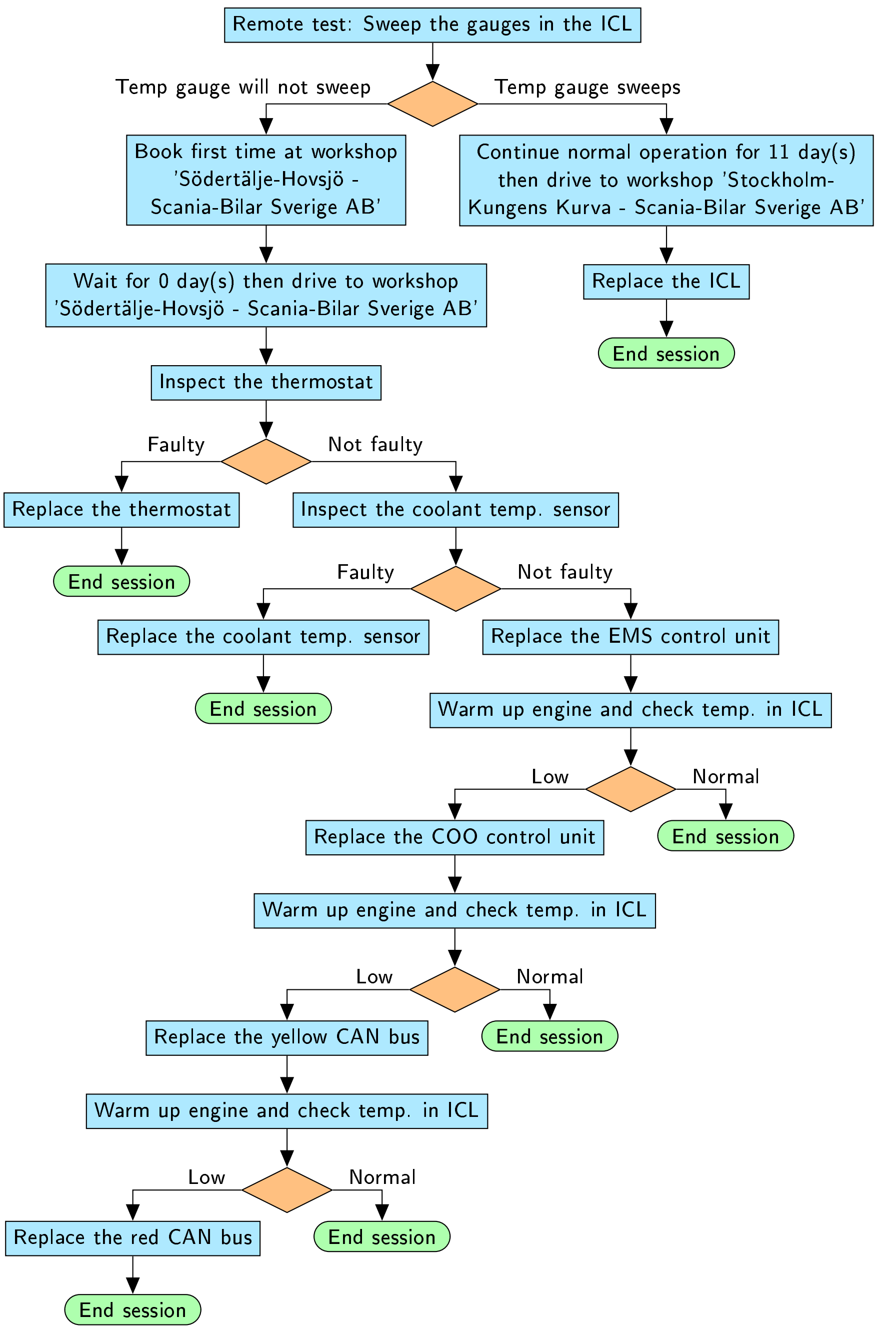
Remote test: Sweep the gauges in the ICL
Temp gauge will not sweep
Temp gauge sweeps
Book first time at workshop 'Södertälje-Hovsjö - Scania-Bilar Sverige AB'
Continue normal operation for 11 day(s) then drive to workshop 'Stockholm-Kungens Kurva - Scania-Bilar Sverige AB'
Wait for 0 day(s) then drive to workshop 'Södertälje-Hovsjö - Scania-Bilar Sverige AB'
Replace the ICL
End session
Inspect the thermostat
Faulty
Not faulty
Replace the thermostat
Inspect the coolant temp. sensor
End session
Faulty
Not faulty
Replace the coolant temp. sensor
Replace the EMS control unit
End session
Warm up engine and check temp. in ICL
Low
Normal
Replace the COO control unit
End session
Warm up engine and check temp. in ICL
Low
Normal
Replace the yellow CAN bus
End session
Warm up engine and check temp. in ICL
Low
Normal
Replace the red CAN bus
End session
End session

Figure 1.4: A full conditional troubleshooting plan for a very small example of troubleshooting a problem with the engine temperature.

more choices and that the time scope of the plans is larger. Instead of a couple of hours, the duration of a single troubleshooting session can span over several weeks. This means that the cost models must take greater consideration to costs related to the downtime of the vehicle as this becomes more important for the decision making.

## 1.3 Outline of Thesis

The thesis is organized into two parts. The first part introduces the topic and gives an overview of the research contributions from all the research done for this thesis. This chapter presented the background and motivation for the research and the different problem formulations considered. The following chapter presents the theoretical background and goes through some related work in the areas of diagnosis and decision making. The third chapter presents and summarizes the main contributions from all publications made for this thesis.

The second part of the thesis consists of a collection of five selected research papers that represent the core of the contributions. These are two published conference papers [6, 9], two published journal papers [8, 11], and one submitted journal manuscript [12].

# Chapter 2

# Theoretical Background

In Section 1.2 we separated the troubleshooting problem into two subproblems, the diagnostic problem and the decision problem. In this chapter we will give an overview of the research areas related to these problem formulations.

## 2.1 Diagnosis

A general approach to the diagnostic problem in the literature is model-based diagnosis where a model is used for correlating the inputs and outputs of the system with the presence of faults. This model can for example explicitly represent the behavior of the system using a model derived from expert knowledge. It can also be a black-box model learned from process data collected from the system.

The major research areas and approaches to model-based diagnosis are discussed below. Of these research areas, it is primarily the probabilistic approaches for diagnosis that are considered in this thesis.

### 2.1.1 Fault Detection and Isolation

The Fault Detection and Isolation (FDI) approach to diagnosis originated in the area of automatic control research. For thorough reviews of the FDI approach, see e.g. [22, 64, 136, 137]. A typical FDI model describes the nominal behavior of the system using a state space model of differential algebraic equations (DAE:s):

$$\begin{aligned} d\mathbf{x}/dt &= f(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) &= g(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \tag{2.1}$$

where the state vector $\mathbf{x}(t)$, the input vector $\mathbf{u}(t)$, and the output vector $\mathbf{y}(t)$ are vectors of continuous variables, and the *behavioral model $f$* and the *observation model $g$* are functions of the input and state vectors [136]. As many laws of physics are DAE:s, this type of model can with high precision capture the behavior of dynamic mechanical, hydraulic, and electrical systems.

To achieve fault detection, a subset of the input and output variables are observed with the observation vector $\mathbf{z}$. If the observation vector makes (2.1) inconsistent, we can conclude that the system is not behaving nominally and there must be a fault present. If the observation vector is such that (2.1) is analytically redundant, i.e. it is possible to find unique solutions for all the unknowns using only a subset of the equations, then it can be possible to further pinpoint the location of the fault by analyzing which subsets of the equations are inconsistent.

A *residual generator* is a model for detecting inconsistencies between the expected and actual behavior of the system using the observation vector as input. The output of the residual generator, the *residual*, is a scalar $r$ such that $r(\mathbf{z}) = 0$ if and only if (2.1) is consistent with regard to $\mathbf{z}$. In practice $r(\mathbf{z})$ will never be zero because of process noise, but this can be overcome by thresholding the value using statistical tests [18]. Two different approaches for creating residual generators are the *parity space approach* [31] and the *structural approach* [14]. For both these approaches different subsets of the equations in the system model are transformed to make the residual generators. If each equation is associated with a component, then each residual generator can be associated with a set of components. When a residual is non-zero, that indicates that there is a fault on at least one of the components associated with that residual generator.

Given an accurate model of the nominal behavior, the FDI methods can become very powerful in detecting faults. It is even possible to detect faults that have not explicitly been modeled because anything that deviates from nominal behavior is considered a fault. When applied to on-board diagnosis of vehicles, the FDI approach has been shown to be successful in detecting even very small sensor and actuator faults [135]. However, creating FDI models requires detailed knowledge of the behavior of the system which can require great effort to obtain.

### 2.1.2 Consistency-Based Diagnosis

In consistency-based diagnosis, the task is to infer which of the system's components have deviating behavior in order to explain inconsistencies between expected and observed system behavior. Consistency-based diagnosis was originally formulated in first-order logic [39, 98], where the model of the system is a triple $\langle SD, COMPS, OBS \rangle$ where:

- *SD* is the *system description*, a set of first-order sentences,
- *COMPS* are the system *components*, a finite set of constants,
- *OBS* are the *observations*, also a set of first-order sentences.

A special predicate *AB* is used such that $AB(c)$ means that component $c$ behaves abnormally, i.e. component $c$ is faulty. In the simplest case, faulty components have arbitrary behavior and only the nominal behavior of the system is modeled. Then each sentence in the system description is on the form:

$$\neg AB(c) \rightarrow \phi$$

where $\phi$ is an arbitrary logical formula describing the behavior of the component $c$ when it is not faulty. Figure 2.1 shows a small logical circuit with three multipliers and two adders commonly used in the literature when explaining consistency-based diagnosis [37]. The model for this circuit is:

$$\begin{aligned}
SD: \quad & \neg AB(M_1) \rightarrow X = A \cdot C \\
& \neg AB(M_2) \rightarrow Y = B \cdot D \\
& \neg AB(M_3) \rightarrow Z = C \cdot R \\
& \neg AB(A_1) \rightarrow F = X + Y \\
& \neg AB(A_2) \rightarrow G = Y + Z \\
COMPS: \quad & M_1, M_2, M_3, A_1, A_2 \\
OBS: \quad & A = 3, B = 2, C = 2, D = 3, E = 3, F = 10, G = 12
\end{aligned}$$

.

In consistency-based diagnosis, a *consistent diagnosis* is a set $\Delta \subseteq COMPS$ of components such that if all components in $\Delta$ are faulty and all others are non-faulty, then the system model is consistent, i.e.

$$SD \cup OBS \cup \{AB(c) : c \in \Delta\} \cup \{\neg AB(c) : c \in COMPS \setminus \Delta\}$$

is consistent.

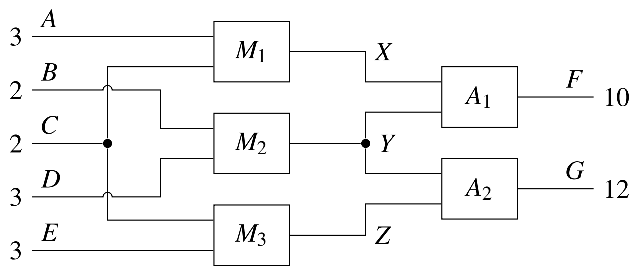

Figure 2.1: A small logical circuit [37].

When only the nominal behavior is modeled, all diagnoses that are supersets of a consistent diagnosis are also consistent. Therefore, one is typically interested in finding only the *minimal diagnoses*, where a minimal diagnosis is a consistent diagnosis $\Delta$ such that there exist no other diagnosis $\Delta' \subset \Delta$ that is a consistent diagnosis.

The common approach to computing the minimal diagnoses is to first identify all *minimal conflict sets*. When only nominal behavior is modeled, a minimal conflict set is a minimal set of components of which at least one component must be faulty in order to explain the observations, i.e. the set of components $C \subseteq COMPS$ is a minimal conflict set if and only if:

$$SD \cup OBS \models \bigvee_{c \in C} AB(c)$$

and for all $C' \subset C$

$$SD \cup OBS \not\models \bigvee_{c \in C'} AB(c).$$

Let $\mathcal{C}$ be the set of all minimal conflict sets. Then each consistent diagnosis $\Delta$ is a hitting set of $\mathcal{C}$. As such $\Delta$ has at least one element in common with every conflict set in $\mathcal{C}$, i.e. for all $C \in \mathcal{C}$: $\Delta \cap C \neq \emptyset$. Given $\mathcal{C}$, the set of all minimal diagnoses can be computed by solving the minimal set covering problem and thereby finding all minimal hitting sets to $\mathcal{C}$. The set of all minimal conflict sets can for example be computed with an Assumption-based Truth Maintenance System [37] and the HS-tree algorithm [98]. The minimal conflict sets for the example are $\{M_1, M_2, A_1\}$ (because the correct answer $F = 12$ is expected unless one of these are faulty) and $\{M_1, M_3, A_1, A_2\}$ (because if $M_1$ and $A_1$ are not faulty then $Y = 4$ and thereby $G = 10$ is expected unless one of $M_3$ and $A_2$ is faulty; on the other hand if $M_3$ and $A_2$ are not faulty then $Y = 6$ and thereby $F = 12$ is expected unless one of $M_1$ and $A_1$ is faulty). The minimal diagnoses are $\{M_1\}$, $\{A_1\}$, $\{M_2, M_3\}$, and $\{M_2, A_2\}$.

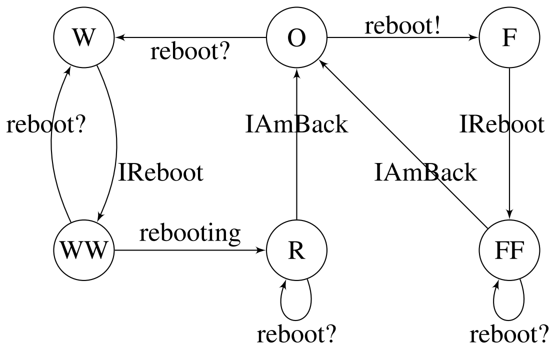

Figure 2.2: Example of a Discrete Event System [54].

It is also possible to have system models where faulty behavior is modeled explicitly [38, 88]. The consistency-based diagnosis approach can also be combined with the FDI approach by creating residual generators such that each non-zero residual maps to a conflict set [14, 33]. Consistency-based methods can then be used to compute consistent diagnoses.

A model for consistency-based diagnosis can be generated from wiring diagrams and other system specifications once models for individual components are created. Consistency-based diagnosis using logical models has been shown to perform well for isolating faults in large real-world applications such as electronic circuits [74].

### 2.1.3 Discrete Event Systems

In the Discrete Event Systems approach, the system to be diagnosed is modeled with a set of states that the system can be in and a set of events that can cause the system to transition between states. Figure 2.2 shows an example of a small Discrete Event System [54]. Some of the events occur due to faults (reboot!) and some, but not all, of the events give rise to observations (IReboot, IAmBack).

The diagnosis task is to estimate which sequences of events that possibly can have occurred, in particular those sequences that include fault events. For example, if the system in Figure 2.2 is known to initially be in state O and we observe the events "IReboot" and "IAmBack", two possible sequences of events are

$$\text{reboot?} \rightarrow \text{IReboot} \rightarrow \text{rebooting} \rightarrow \text{IAmBack}$$

which contain no fault events, and

$$\text{reboot!} \rightarrow \text{IReboot} \rightarrow \text{IAmBack}$$

which contain the fault event "reboot!".

Discrete Event Systems are particularly suited for modeling large distributed systems such as communications networks where no single entity has access to all information. Each node in the distributed systems can be modeled with its own set of states and messages sent between the nodes are modeled as events. Common modeling and solution methods for Discrete Event Systems are based on Petri Nets [51] and state automata [95, 149].

### 2.1.4 Data-Driven Methods

Data-driven methods for diagnosis rely primarily on data collected from the system. The model that is used is learned from this data and it does not necessarily need to represent any real physical relations of the system. Using methods from the machine learning field, the goal is typically to output the single diagnosis that best matches the observed data. It is a two-step process: first a model is learned from large amounts of data and then the model is used to compute diagnoses.

Learning can be *supervised* or *unsupervised*, and for diagnosis, primarily supervised learning methods have been employed.

In supervised learning, the data that is used for learning the model is labeled with the true diagnosis of the system from which it was collected. The data $D = \{(x_1, y_1) \dots (x_n, y_n)\}$ consists of examples $(x_i, y_i)$ where $x_i$ is a set of observed values of *features* of the system (e.g. signal values and symptoms) and $y_i$ is the *label* (e.g. the diagnosis at the time). The model is a function that maps the outcome space of the features to labels and it is is learned from the data using some machine-learning method. Once learned, this function can be used to classify new examples where only the values of the features are known. Supervised learning methods that have been applied for diagnosis include *Support Vector Machines* [34, 65, 107, 110] where different classes are separated with a hyperplane with maximal margin (see Figure 2.3), *Decision Forests* where several decision trees vote on the class [123], *Neural Networks* where a network of sigmoid functions is learned to fit the data [66, 71, 147], and *Case-Based Reasoning* where classification is done by identifying similar examples in the data [16, 77, 147].

In unsupervised learning the data is collected without knowledge of the system's state, i.e. $D = \{x_1, \dots, x_n\}$. When unsupervised learning has been applied to diagnosis, the unlabeled data is clustered to detect anomalies which can be due to faults or impending faults [134].

A benefit with data-driven methods is that there is no need for extensive knowledge engineering. However, to learn complex dependencies large quantities of data is needed which can be hard to obtain, especially for infrequent faults.

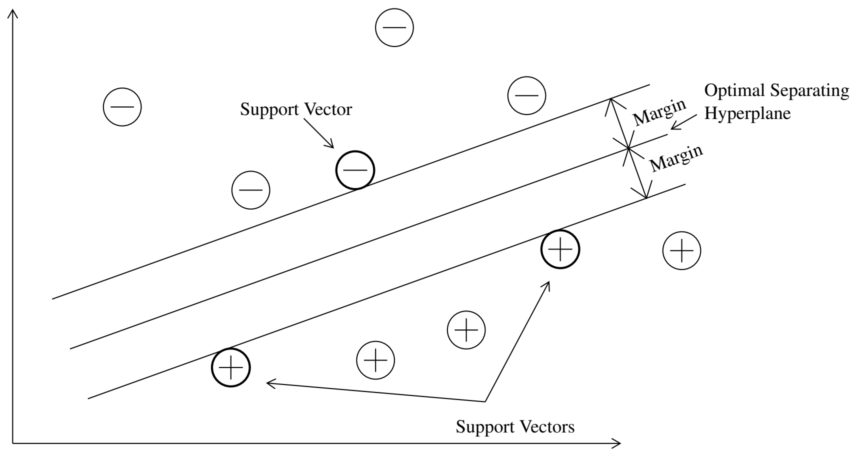

Figure 2.3: Example of a linear Support Vector Machine for a two-dimensional feature space. The two classes negative (-) and positive (+) are separated with a line with maximal margin [110].

## 2.1.5 Probabilistic Approaches

Probabilistic approaches for diagnosis are needed when the goal is to compute the probabilities of diagnoses or faults (equations (1.1) and (1.2)).

**Example 2.1.** Figure 2.4 shows a diagram of a very simplified common rail fuel injection system with two fuel injectors connected to a pressurized fuel rail. In this example, there can be faults on the fuel pump and the fuel injectors. Any of these faults can cause the fuel pressure in the fuel rail to become low. When this pressure is low, this can cause the output effect in the cylinders to become low. A faulty fuel injector can also directly cause the cylinder output effect to become low. The behavior of the faults can vary depending on the severity and nature of the faults.

A common approach to modeling dependencies between faults and observations that are probabilistic is by using a graphical model such as Bayesian networks [93]. The nodes in a Bayesian network are random variables and the edges are probabilistic dependencies such that each random variable $X_i$ is conditionally independent given its parents $pa(X_i)$ in the graph:

$$p(X_1,\ldots,X_n) = \prod_{i=1}^{n} p(X_i|pa(X_i)).$$

Each variable $X_i$ is associated with a conditional probability distribution (CPD) $p(X_i|pa(X_i))$. If the variables are discrete random variables the CPD is sometimes called a conditional probability table (CPT). Figure 2.5 shows an example of a

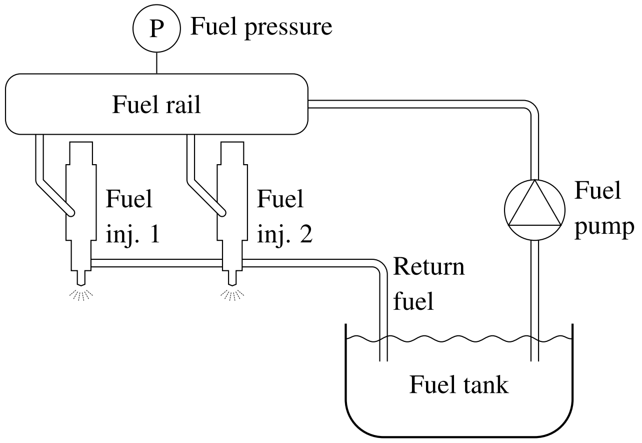

Figure 2.4: Diagram of a simplified common rail fuel injection system.

Bayesian network graph for the system in Example 2.1 with binary variables $F_{inj.1}$, $F_{inj.2}$, $F_{pump}$, $O_{pres.}$, $O_{eff.1}$, and $O_{eff.1}$. In this model the faults are assumed to occur independently and there is an edge to each observable symptom from its causes.

In the context of diagnosis, the Bayesian network model provides CPD:s that describe the behavior of observable properties of the system given its diagnosis. The *prior distribution* is the joint probability distribution over the possible diagnoses, in this example the distribution $p(F_{inj.1}, F_{inj.2}, F_{pump})$. The *likelihood* is the probability of making a specific observations given a known diagnosis, e.g. the probability of observing the fuel pressure being low given that we know whether the fuel injectors and the fuel pump are faulty or not:

$$\Pr(O_{pres.} = \text{low} | F_{inj.1}, F_{inj.2}, F_{pump}).$$

The *posterior distribution* is the probability distribution of the diagnoses given known observations, e.g.:

$$p(F_{inj.1}, F_{inj.2}, F_{pump} | O_{pres.}).$$

This is computed by the diagnoser in a process called *inference*. Sometimes only the marginalized fault probabilities are needed. Then inference would be to compute, e.g.:

$$\Pr(F_i = \text{faulty} | O_{pres.}) \text{ for } i = inj.1, inj.2, pump.$$

Many inference algorithms rely on *Bayes' rule* to compute posterior probabilities from known likelihoods and priors:

$$p(\text{posterior}) \propto \Pr(\text{likelihood}) p(\text{prior}).$$

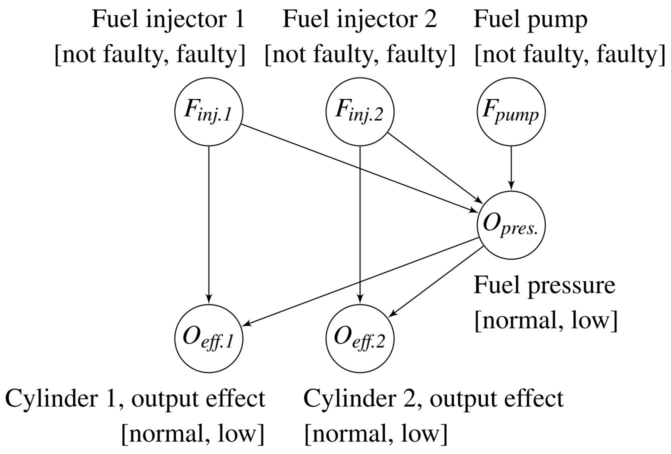

Figure 2.5: Bayesian network for diagnosis of the system described in Example 2.1.

Exact inference using even the most efficient inference algorithms, e.g. the Junction Tree algorithm [76], is in general intractable [32]. Faster inference can be achieved if the network has special structure, e.g. by being singly connected [92] or only having CPT:s that are Noisy-Or distributions [61, 63]. Faster inference can also be achieved by performing approximate inference, e.g. by sampling [148].

Existing probabilistic approaches for diagnosis vary in how modeling and inference is made. A *dynamic* Bayesian network [41] can be used to model systems that vary over time. A Bayesian network that is not dynamic is said to be *static*. When static Bayesian networks are used the observation variables are often symptoms and test results rather than direct sensor values, e.g. "Fuel pressure is lower than expected" [27, 62, 72, 99, 139]. In distinction to the FDI methods for diagnosis, the static model is incapable of simulating the system and less efficient for fault detection. Instead static Bayesian networks are often used in applications of sequential decision making and troubleshooting when a fault has already been detected [62, 67, 75]. A diagnostic system based on the FDI approach can be combined with a Bayesian network based diagnoser by using the residuals as observations in the static network [139].

Dynamic Bayesian networks can be used when the model is created using a physical description of the system such as a bond-graph [103, 104, 105] or residual equations [119, 145]. Non-stationary Dynamic Bayesian networks [102] can be used to model event driven troubleshooting processes [3].

Bayesian network models for diagnosis can be created manually by experts using a modeling tool, e.g. [42, 68, 80, 124]. There are also methods for automatic generation of diagnostic Bayesian networks from other sources common in the industry, such as Failure Mode and Effects Analysis (FMEA) [48] and Fault Tree Analysis [23]. Models such as Object-Oriented Bayesian networks can be used to simplify modeling when many systems with shared structure are modeled [70].

It is also possible to use data-driven methods to learn Bayesian networks from data, either both the structure and parameters or parameters only, see e.g. [49, 96, 122]. If data is scarce, it is possible to combine model learning from data with manual model creation where the data is used to tune model parameters whose initial values are provided by experts [96]. When a Bayesian approach to model learning is used, the original model parameters $\theta$ are associated with a probability distribution $p(\theta)$ that describe our uncertainty in their correctness. If we have collected data $D$ from the system and know its likelihood given the parameters, we can compute a new probability distribution $p(\theta|D)$ which takes into account the knowledge of this data:

$$p(\theta|D) \propto \Pr(D|\theta)p(\theta).$$

This distribution can for example be computed using conjugate priors for certain specific model types. For more general models it is possible to use sampling techniques such as Gibbs sampling [50] and the Metropolis-Hastings algorithm [60].

There are other probabilistic methods for diagnosis that do not rely on using Bayesian networks. Some of these methods are related to the FDI approach where the model is a probabilistic state space model where in (2.1) $f$ and $g$ are instead probability distributions. If the model is linear and the distributions are normal distributions the *Kalman filter* [69] can be used for exact inference, see e.g. [57]. For general models and distributions it is common to use a particle filter [53], where the posterior distribution is estimated with particles representing possible system states which are weighted with their likelihood, see e.g. [36, 86, 138].

Another probabilistic approach is based on consistency-based diagnosis. First the set of minimal diagnoses is computed as described in Section 2.1.2. Then the probabilities of diagnoses are estimated by letting the probabilities of inconsistent diagnoses be zero and letting the probabilities of consistent diagnoses be proportional to their prior probabilities [37].

## 2.2 Decision Making

In Section 1.2 we formulated the decision problem to be about deciding which action to perform next when troubleshooting the system. In this section we will discuss three existing approaches to decision making in the context of troubleshooting:

- the *decision theoretic* approach where alternative decisions are evaluated heuristically based on their expected utility without necessarily considering all future decisions,

- the *automated planning* approach where a decision is made by creating and evaluating a complete or almost complete plan for solving the troubleshooting problem, and
- the *case-based reasoning* approach where decisions are made based on previous experience of which decisions have been taken before in similar situations.

We will emphasize the decision theoretic and automated planning approaches as it are these approaches that are used in the thesis.

To explain the approaches we will use a very small example problem:

**Example 2.2.** Consider the troubleshooting problem described in Example 2.1. Assume that there are three possible troubleshooting actions: *replace injectors* (both injectors are replaced at the same time), *replace fuel pump*, and *check rail pressure* (pressure is low if and only if faults are present). Let the cost of these action be 90, 40, and 10 respectively. At the onset of troubleshooting there is a single fault. There is a 25% chance this is the fuel pump and a 75% chance that this is one of the injectors.

When there is a finite number of actions and each action has a finite number of outcomes, a naive method for finding the optimal decision is by using a *decision tree*. The decision tree shows every possible sequence of actions that leads to a solution of the problem together with the costs of the actions and the probability of possible actions.

A decision tree for Example 2.2 is shown in Figure 2.6. The decision tree has three types of nodes: decision nodes (squares), chance nodes (circles), and end nodes (triangles). At a decision node a decision can be made by selecting one of the branches where each branch corresponds to a possible action. If the action has multiple outcomes, it is followed by a chance node whose outgoing branches correspond to the possible outcomes of the action. The end nodes are labeled with the total cost and probability of reaching there from the root node. The decision maker can then use the tree to select the decisions that give the most favorable set of outcomes. For example, choosing to first replace the injectors then check whether this solved the problem, and if the pressure is still low replace the fuel pump, gives a 25% chance of a total troubleshooting cost of 140 and a 75 % chance of a total troubleshooting cost of 100, thus an expected cost of repair of 110 which is optimal in this example. In the decision tree in Figure 2.6 we have omitted actions that do not advance the troubleshooting, e.g. pressure check when we already know the pressure and replacements of already replaced components.

Decision trees have been used for many types of decision problems, especially in the areas of economics and game-theory [106]. However when there are more alternative actions and the sequences of decisions needed to reach an end node are

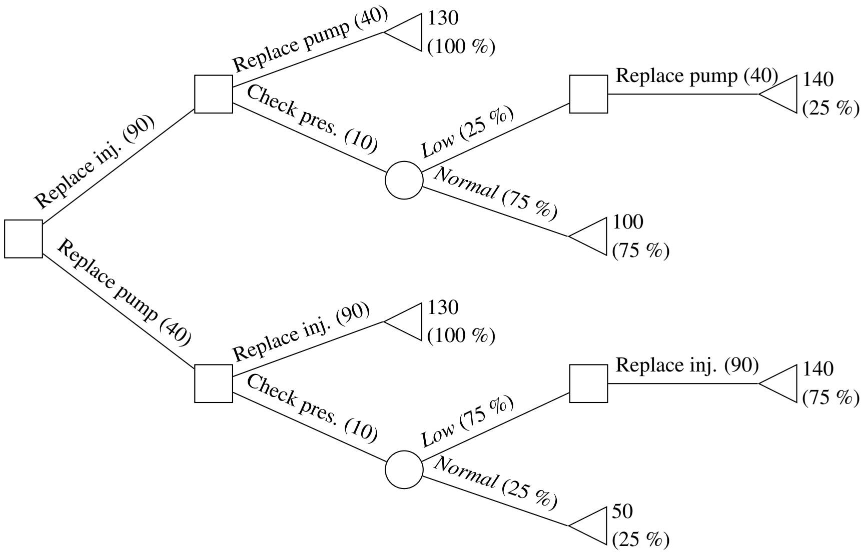

Figure 2.6: A decision tree for Example 2.2. Decision nodes are shown with squares, chance nodes with circles, and end nodes with triangles.

longer, the decision tree method obviously becomes intractable as the decision tree becomes immensely large. We will use it here for Example 2.2 with the purpose of explaining other approaches to decision making.

### 2.2.1 The Decision-Theoretic Approach

The common decision-theoretic approach to making the decision problem tractable is to prune the decision tree at a certain depth $k$. Depending on whether we want to minimize the expected cost or maximize the expected utility of the tree, a heuristic function is used to estimate the minimal/maximal expected cost/utility of the remaining tree below the pruned branch. Without loss of generality we will assume that the goal is always to minimize the cost rather than to maximize the utility as this is typically the case in the context of troubleshooting. A decision can be computed from the pruned tree by recursively selecting the action that minimizes/maximizes the expected cost/utility for each decision node. This is sometimes referred to as $k$-step look-ahead search [109].

Assume that a state $s$ can be used to describe everything we know of which actions have been performed and which observations have been seen (note that this state is not the same as the actual state of the system, which is unknown). The

$k$-step look-ahead search decision for state $s$ is the action $a^*(s)$ where:

$$a^*(s) = \arg\min_{a \in \mathcal{A}} \Big( cost(a,s) + \sum_{o \in \Omega_a} \Pr(o|a,s) c_{k-1}(s_{o,a}) \Big) \tag{2.2}$$

where $\mathcal{A}$ is the set of all possible actions that we can choose from, $cost(a,s)$ is the cost of performing action $a$ in state $s$, $\Pr(o|a,s)$ is the probability that the action $a$ has the outcome $o$ in state $s$ out of the possible outcomes $\Omega_a$, $s_{o,a}$ is the state of the system after the action $a$ with outcome $o$ has been performed in state $s$, and $c_k$ is the $k$-step look-ahead cost:

$$c_k(s) = \begin{cases} \min_{a \in \mathcal{A}} \Big( cost(a,s) + \sum_{o \in \Omega_a} \Pr(o|a,s) c_{k-1}(s_{o,a}) \Big) & \text{if } k > 0 \\ h(s) & \text{if } k = 0, \end{cases}$$

where $h$ is a real-valued heuristic function. Typically the value of $k$ is 1, but different decision-theoretic approaches differ on how the outcome probabilities and the heuristic function are computed.

If the diagnostic approach that is used is based on Bayesian networks, then it is straightforward to compute the outcome probabilities. However, when a probabilistic consistency-based diagnostic method as described in Section 2.1.5 is used, then the outcome probability is not readily available. One solution is then to assume that all outcomes that are consistent with the model and the current observations are equally likely [37].

When only observing actions are considered, it is common to use a heuristic based on the information entropy of the probability distribution over faults and diagnoses, see e.g. [37, 46, 101, 150]. The information entropy of the probability distribution of a random variable is a measure of how much more information is needed to learn the true value of the variable [55]. Let $\Pr(d|s)$ be the probability of having the diagnosis $d$ given state $s$. Then the entropy of $s$ is:

$$H(s) = -\sum_d \Pr(d|s) \log \Pr(d|s).$$

Look-ahead search with $k = 1$ and a heuristic based on information entropy has empirically been shown to be remarkably efficient in finding sequences of observing actions that find the true diagnosis at a low expected cost [40]. By directly computing the conditional entropy of the state given a tentative action, it is possible to completely avoid the need of computing outcome probabilities when $k = 1$ [150].

By weighting together the entropy heuristic with other heuristics it is possible to consider decision problems that also include repair actions. Sun and Weld [132] defines such a heuristic:

$$h_{sun}(s) = c_H H(s) + \sum_d \Pr(d|s) c_{rep}(d) \tag{2.3}$$

where $c_H$ is a weighting constant for the entropy, and $c_{rep}(d)$ is the cost of repairing all faults in $d$.

If we would apply one-step look-ahead search together with this heuristic together to the problem in Example 2.2 using $c_H = 10$ (average cost of all observing actions), then in the initial state $s_0$ the decision would be to first replace the injectors using the action $a_{inj}$, because the expected cost of this decision is in (2.2) evaluated to:

$$cost(a_{inj.}, s_0) + \sum_{o \in \Omega_{a_{inj}}} \Pr(o|a_{inj}, s_0) h_{sun}(s_{o,a_{inj}}) = \\ 90 - 10(0.75 \log 0.75 + 0.25 \log 0.25) + 0.25 \cdot 40 \approx 108.1$$

compared to when the fuel pump is replaced using the action $a_{pump}$:

$$cost(a_{pump}, s_0) + \sum_{o \in \Omega_{a_{pump}}} \Pr(o|a_{pump}, s_0) h_{sun}(s_{o,a_{pump}}) = \\ 40 - 10(0.25 \log 0.25 + 0.75 \log 0.75) + 0.75 \cdot 90 \approx 115.6.$$

Heckerman et al. [62] describes another heuristic for troubleshooting that is derived from the exact solution of an alternate simpler problem. In this simpler problem, the set of available actions is reduced to only include repair actions (that remove a specific fault and that are immediately followed by a system check that can determine whether the system is fault free or not) and inspection actions (that deterministically determine whether a specific fault is present or not). It is also assumed that there is exactly one fault present and that action costs are state independent. For this simpler problem, it is possible to efficiently compute the optimal expected cost of repair analytically by considering a troubleshooting strategy where each fault is inspected in a particular order and repairs are made whenever a fault is found to be present. If this order is such that:

$$\frac{p_i}{C_i^o} \geq \frac{p_{i+1}}{C_{i+1}^o}$$

where $p_i$ is the probability of the $i$th fault being present and $C_i^o$ is the inspection cost for this fault, then this troubleshooting strategy is optimal for the simplified problem. For a system with $n$ possible faults the minimal expected cost of repair using this troubleshooting strategy is:

$$ECR^*_{heck}(s) = \sum_{i=1}^{n} \left( \left( 1 - \sum_{j=1}^{i-1} p_j \right) C_i^o + p_i C_i^r \right) \tag{2.4}$$

where $C_i^r$ is the repair cost for fault $i$. If there exists an inspection action for a fault $i$, then $C_i^o$ is the cost of that action and $C_i^r$ is the cost of the repair action plus the cost of the system check action. If there is no inspection action for this fault, the inspection is made by first repairing the fault and then performing the system check. In this case $C_i^o$ is the cost of the repair action plus the system check and $C_i^r = 0$.

In practice there are not many troubleshooting problems that are like this simplified problem. However if we define a heuristic

$$h_{heck}(s) = ECR^*_{heck}(s), \tag{2.5}$$

we get a heuristic that has been shown to be useful together with look-ahead search on more complex decisions problems, see e.g. [62, 75]. There are also extensions to this approach where the action costs can depend on the state and repair actions are not immediately followed by a system check [89, 90].

In Example 2.2 there are no inspection actions and the observation action qualifies as a system check so the inspection cost is 100 for the injectors and 50 for the fuel pump. In the initial state, the ratio is 0.0075 for the injectors and 0.005 for the fuel pump meaning that the injectors should be replaced first. The heuristic value in the initial state is $100 + 0.25 \cdot 50 = 112.5$.

## 2.2.2 The Automated Planning Approach

In the automated planning approach for decision making, the decisions are made by creating and evaluating a complete or almost complete plan for solving the troubleshooting problem. If needed by the user, the entire plan can be provided to the user immediately. Otherwise only the first action in the plan can be shown to the user as in the decision-theoretic approach.

Because the state of the system is unknown and observing actions can have multiple outcomes which we only know the probabilities of, the planning problem is probabilistic and partially observable. These types of problems can be modeled with Partially Observable Markov Decision Processes (POMDP:s) which is a general mathematical framework for sequential decision processes with incomplete knowledge (see e.g. [28]).

### POMDP:s

A discrete POMDP can be described with a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, t, p, c, \gamma \rangle$ where:

- $\mathcal{S}$ is the *state space*, a finite set of states the system can be in,
- $\mathcal{A}$ is the *action space*, a finite set of actions that can be performed,
- $\mathcal{O}$ is the *observation space*, a finite set of observations that can be seen after performing an action,

- $t : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$ is the *transition function*, a probability distribution such that $t(s,a,s')$ is the probability that state $s'$ is reached if action $a$ is performed in state $s$,
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{O} \mapsto [0,1]$ is the *observation function*, a probability distribution such that $p(s,a,s',o)$ is the probability of making observation $o$ when state $s'$ is reached by performing action $a$ in state $s$,
- $c : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the *cost function* where $c(s,a)$ is the cost of performing action $a$ in state $s$ (in the typical definition of POMDP:s this is a reward function $r$, but in the context of troubleshooting a cost function is more suitable), and
- $\gamma \in [0,1]$ is the *discount factor*, a parameter for weighting the costs of future actions.

**Example 2.3.** A POMDP for the troubleshooting problem in Example 2.2 can be defined as follows:

| | |
|---|---|
| $\mathcal{S} = \{s_0, s_1, s_2\}$ | where in state $s_0$ the system has no faults, in state $s_1$ one of the injectors is faulty, and in state $s_2$ the fuel pump is faulty. |
| $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ | where action $a_0$ is a no-op action signaling the end of the troubleshooting session, action $a_1$ replaces the injectors, action $a_2$ replaces the fuel pump, and action $a_3$ observes the fuel pressure. |
| $\mathcal{O} = \{o_0, o_1\}$ | where, if action $a_3$ is performed, $o_0$ is the observation that the fuel pressure is nominal and $o_1$ is the observation that it is low. For all other actions, the observations have no particular meaning and $o_0$ is observed by default. |

$$t(s,a,s') = \begin{cases} 1 & \text{if } a \in \{a_0, a_3\} \text{ and } s = s', \\ 1 & \text{if } a = a_1 \text{ and } (s,s') \in \{(s_0,s_0),(s_1,s_0),(s_2,s_2)\}, \\ 1 & \text{if } a = a_2 \text{ and } (s,s') \in \{(s_0,s_0),(s_1,s_1),(s_2,s_0)\}, \\ 0 & \text{otherwise.} \end{cases}$$

$$p(s,a,s',o) = \begin{cases} 1 - p(s,a,s',o_1) & \text{if } o = o_0, \\ 1 & \text{if } a = a_3 \text{ and } s \in \{s_1, s_2\}, \\ 0 & \text{otherwise.} \end{cases}$$

$$c(s,a) = \begin{cases} 0 & \text{if } a = a_0 \text{ and } s = s_0, \\ 90 & \text{if } a = a_1 \\ 40 & \text{if } a = a_2 \\ 10 & \text{if } a = a_3 \\ \infty & \text{otherwise.} \end{cases}$$

$$\gamma = 1.$$

Because the true state is not known, our belief regarding which this state is is represented with a *belief state*. A belief state is a probability distribution over $\mathcal{S}$ such that $b(s)$ is the probability that state $s$ is the true state. The *belief state space* $\mathcal{B}$ is the set of all possible belief states. By performing actions and making observations we evolve the belief state. When an action $a$ is performed in a belief state $b$ and the observation $o$ is seen, a new belief state $b_{a,o}$ is reached where:

$$b_{a,o}(s') = \frac{\sum_{s \in \mathcal{S}} p(s,a,s',o)t(s,a,s')b(s)}{\bar{p}(b,a,o)} \quad \text{for all } s' \in \mathcal{S}, \tag{2.6}$$

and

$$\bar{p}(b,a,o) = \sum_{s,s'' \in \mathcal{S}} p(s,a,s'',o)t(s,a,s'')b(s) \tag{2.7}$$

is the belief state transition probability for observation outcome $o$ of $a$ in $b$.

Problems formulated as POMDP:s are usually solved by finding an optimal or suboptimal *policy*, which is a function $\pi : \mathcal{B} \mapsto \mathcal{A}$ that maps every belief state to an action. The expected cost of a policy $\pi$ is given by a *value function* $V_\pi : \mathcal{B} \mapsto \mathbb{R}$ where:

$$V_\pi(b) = \bar{c}(b,a) + \gamma \sum_{o \in \mathcal{O}} \bar{p}(b,a,o)V(b_{a,o}), \tag{2.8}$$

where $a = \pi(b)$, and $\bar{c}(b,a) = \sum_{s \in \mathcal{S}} b(s)c(s,a)$. A plan can be obtained from a policy by applying the policy to all belief states that are *reachable* from a given initial belief state. A reachable belief state is a belief state that can be reached from the initial belief state by multiple applications of (2.6) with a non-zero belief state transition probability.

An *optimal policy* is a policy $\pi^*$ such that

$$\pi^*(b) = \operatorname*{arg\,min}_{a \in \mathcal{A}} \Big( \bar{c}(b,a) + \gamma \sum_{o \in \mathcal{O}} \bar{p}(b,a,o)\pi^*(b_{a,o}) \Big). \tag{2.9}$$

When the discount factor $\gamma < 1$, the influence of future actions on the expected cost decreases exponentially with the number of steps into the future and the expected cost of any policy is finite. However when $\gamma = 1$, $c$ is non-negative, and there exist *goal states* for which the optimal expected cost is zero, the expected cost is only finite for policies guaranteed to reach a goal state. Then the POMDP is said to be a *goal POMDP* [24]. By identifying (2.8) and (2.9) with (1.5) and (1.6), we see that the troubleshooting problem as we defined it in Section 1.2 is a goal POMDP.

It is computationally intractable to compute an optimal policy for all but trivially small POMDP:s [28]. However, there are many approximate algorithms capable of computing good suboptimal policies to larger POMDPs. Notable among these are the algorithms based on Point-Based Value Iteration [97] (see e.g. [73, 121, 125, 126, 129]). Many of these use knowledge of a known initial belief state $b_0$ and focus on finding policies that are good in the reachable belief state space.

Because the belief state space is continuous, it is difficult to efficiently represent policies. However the optimal value function is concave (or convex if a reward function is used) and can be approximated with arbitrary precision by taking the minimum of a set of $|\mathcal{S}|$-dimensional hyperplanes [128]. The point-based algorithms represent these hyperplanes with a set of vectors $\Gamma$ called $\alpha$-vectors and each $\alpha$-vector $\alpha \in \Gamma$ is associated with an action $action(\alpha)$. The set $\Gamma$ defines a policy $\pi_\Gamma$ where $\pi_\Gamma(b) = action(\alpha_b^*)$ where $\alpha_b^*$ is the $\alpha$-vector that has the smallest value in $b$:

$$\alpha_b^* = \arg\min_{\alpha \in \Gamma} \sum_{s \in \mathcal{S}} b(s)\alpha(s).$$

The value function of $\pi_\Gamma$ is approximated by the function $V_\Gamma$ where

$$V_\Gamma(b) = \sum_{s \in \mathcal{S}} b(s)\alpha_b^*(s).$$

The initial set of $\alpha$-vectors in $\Gamma$ can for example be obtained by creating one $\alpha$-vector for every action and letting $\alpha(s)$ be the cost of performing $action(\alpha)$ repeatedly. The suboptimal policy $\pi_\Gamma$ is then incrementally improved by performing so called *backups* on selected belief states. When a belief state $b$ is backed up, a new $\alpha$-vector is added to $\Gamma$:

$$\Gamma' = \Gamma \cup \{\arg\min_{\beta_a : a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b(s)\beta_a(s)\} \tag{2.10}$$

where each $\beta_a$ is an $\alpha$-vector associated with the action $a$ and for all $s \in \mathcal{S}$:

$$\beta_a(s) = c(s,a) + \gamma \sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} p(s,a,s',o)t(s,a,s')\beta_{a,o}(s') \tag{2.11}$$

and $\beta_{a,o}$ are $\alpha$-vectors for each action $a \in \mathcal{A}$ and observation $o \in \mathcal{O}$ corresponding to the best policy in the next belief state $b_{a,o}$:

$$\beta_{a,o} = \arg\min_{\alpha \in \Gamma} \sum_{s \in \mathcal{S}} b_{a,o}(s)\alpha(s).$$

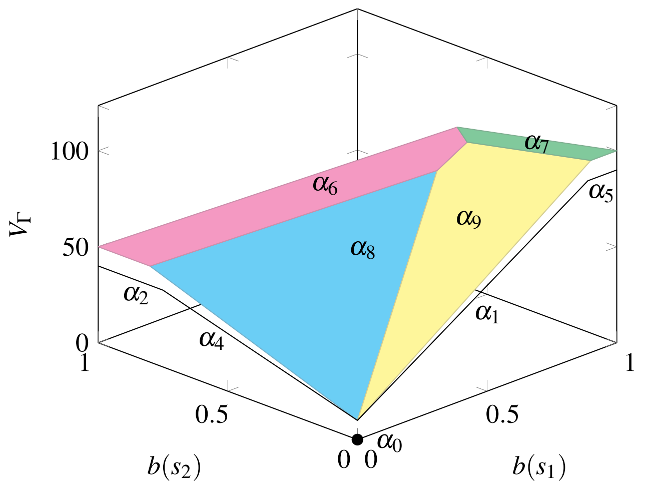

Figure 2.7: The value of $V_\Gamma(b)$ for all $b \in \mathcal{B}$ for Example 2.4 forms a concave surface.

**Example 2.4.** Consider the POMDP model in Example 2.3. Let the set of $\alpha$-vectors be initialized to $\Gamma = \{\alpha_0, \alpha_1, \alpha_2\}$ where $\alpha_0 = (0, \infty, \infty)$, $\alpha_1 = (90, 90, \infty)$, $\alpha_2 = (40, \infty, 40)$, and for $i = 0, 1, 2$, $action(\alpha_i) = a_i$. When backing up an arbitrary selected belief state $b = (0.0, 0.5, 0.5)$ by applying (2.10) on $b$, we evaluate (2.11) for each action and $\beta_{a_0} = (0, \infty, \infty)$, $\beta_{a_1} = (130, 130, 130)$, $\beta_{a_2} = (130, 130, 130)$, $\beta_{a_3} = (10, \infty, \infty)$ of which $\beta_{a_1}$ is minimal in the $b$ and therefore $\alpha_3 = \beta_{a_1}$ is added to $\Gamma$. Further applications of (2.10) on arbitrary selected belief states yields:

| $i$ | $b$ | $\alpha_i$ | $action(\alpha_i)$ |
|---|---|---|---|
| 4 | $(0.5, 0, 0.5)$ | $(10, \infty, 50)$ | $a_3$ |
| 5 | $(0.5, 0.5, 0)$ | $(10, 100, \infty)$ | $a_3$ |
| 6 | $(0, 0.2, 0.8)$ | $(50, 140, 50)$ | $a_2$ |
| 7 | $(0, 0.8, 0.2)$ | $(100, 100, 140)$ | $a_1$ |
| 8 | $(0.5, 0.1, 0.4)$ | $(10, 150, 60)$ | $a_3$ |
| 9 | $(0.5, 0.4, 0.1)$ | $(10, 110, 150)$ | $a_3$ |

Figure 2.7 shows the value of $V_\Gamma(b)$ for all $b \in \mathcal{B}$ for Example 2.4. New facets are formed when $\alpha$-vectors that are minimal for some region of the $\mathcal{B}$ are added to $\Gamma$. The vectors $\alpha_0$, $\alpha_1$, $\alpha_2$, $\alpha_4$, and $\alpha_5$ are only finite for a subspace of $\mathcal{B}$ and therefore shown only with a point or a line. The vector $\alpha_3$ is dominated everywhere by other $\alpha$-vectors and does therefore not appear in the graph.

The backup operation is central to all point-based POMDP solvers, but they vary on how $\alpha$-vectors are initialized and how belief states are selected for backups. If the belief states are selected appropriately, $V_\Gamma(b_0)$ converges toward the optimal value $V_{\pi^*}(b_0)$ as the number of backups goes to infinity [125]. Also if the initial set of $\alpha$-vectors are initialized so that $V_\Gamma(b)$ is greater than or equal to the optimal value, $V_\Gamma(b)$ will remain greater than or equal to the optimal value after backups. The point-based algorithms are *anytime* algorithms, which means that they can at any time return the currently best policy found. This is a useful property in real-time applications. However, the computational complexity of the POMDP is not avoided. The size of $\Gamma$ increases as more backups are performed causing the backup operation to become more and more computationally expensive. To keep the size of $\Gamma$ down many algorithms include a pruning step to remove unneeded $\alpha$-vectors such as the vector $\alpha_3$ in Example 2.4.

It is sometimes possible to represent a POMDP as a *factored* POMDP model where the states are represented with sets of random variables and the transition and observation functions are represented as Bayesian networks [25]. Then properties such as conditional independence between the variables can be exploited for more efficient representation of belief states and backup operations. Further, if certain state variables are fully observable, POMDP models such as Mixed Observability Markov Decision Processes can leverage this [13].

There are not many existing applications of POMDP:s for decision making in diagnosis and troubleshooting. Benazera and Chanthery [20] discusses potential applications of POMDPs for repair and maintenance planning and Williams [146] uses a POMDP for a spoken dialog troubleshooting system.

### Belief MDP:s

When the discount factor $\gamma = 1$, the performance of many of the point-based methods breaks down. It is possible to transform a POMDP into a continuous fully observable Markov Decision Process (MDP) by replacing the state space with the belief state space [28]. This is called a *belief MDP*, and for a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, t, p, c, \gamma \rangle$, the belief MDP is a tuple $\langle \mathcal{B}, \mathcal{A}, \tau, \bar{c}, \gamma \rangle$ where:

- $\mathcal{B}$ is the state space, the previous belief state space, an $|\mathcal{S}|$-dimensional hypercube,
- $\mathcal{A}$ is the action space, same as before,
- $\tau : \mathcal{B} \times \mathcal{A} \times \mathcal{B} \mapsto [0,1]$ is the transition function, where

  $$\tau(b,a,b') = \begin{cases} \bar{p}(b,a,o) & \text{if } b' = b_{a,o} \text{ for some } o \in \mathcal{O}, \\ 0 & \text{otherwise,} \end{cases}$$

  and $b_{a,o}$ and $\bar{p}(b,a,o)$ are given by (2.6) and (2.7).
- $\bar{c} : \mathcal{B} \times \mathcal{A} \mapsto \mathbb{R}$ is the cost function where $\bar{c}(b,a) = \sum_{s \in \mathcal{S}} b(s)c(s,a)$.

Policies for belief MDP:s can be computed using ordinary methods for MDP:s that are capable of handling state spaces of infinite size. When a goal POMDP is transformed into a belief MDP and there is a known initial belief state $b_0$, the MDP is sometimes called a *Stochastic Shortest Path Problem* (SSPP) [21]. Problems formulated as goal POMDP:s can often be solved more efficiently as belief MDP:s using ordinary MDP solution methods [24].

A large family of algorithms for computing near-optimal policies for these types of MDP:s are derivatives of the *Real Time Dynamic Programming* (RTDP) algorithm [17]. The RTDP algorithm computes a partial policy $\pi$ and estimates its expected cost with a value function $f$ that is defined for an enumerated subset of the state space. It explores the state space randomly through a series of depth first random walks starting from the initial state. Such a random walk is called a *trial* and it is a recursive function consisting of two steps:

1. A *backup* is performed on the current state $b$ by doing a Bellman update [19]:

$$f(b) \leftarrow \min_{a \in \mathcal{S}} \Big( \bar{c}(b,a) + \gamma \sum_{b' \in succ(a,b)} \tau(b,a,b') f(b') \Big) \tag{2.12}$$

$$\pi(b) \leftarrow \arg\min_{a \in \mathcal{S}} \Big( \bar{c}(b,a) + \gamma \sum_{b' \in succ(a,b)} \tau(b,a,b') f(b') \Big) \tag{2.13}$$

   where $succ(a,b) = \{b' \in \mathcal{B} : \tau(b,a,b') > 0\}$ is the set of successor states of state $b$ given action $a$.

2. Unless a stopping condition is met, the function is called again using a randomly selected state in $succ(\pi(b),b)$ as the current state. A trial can be stopped if e.g. $f(b) = 0$, a certain recursion depth is achieved, or a timeout occurs.

Because the belief state space is continuous and the policy is defined over a finite set of states, a complete policy is never created. However, when used in a real-time setting where the policy is recomputed after every decision, the complete policy is not needed. If the value function is initialized to a value lower than the optimal expected cost and any state that is reachable from the initial state using the current policy is backed up with a non-zero probability, then the value function and policy will converge towards the optimal ones in the initial state [17]. RTDP-based algorithms benefit from using strong heuristics when initializing the value function. Many also use two value functions giving two-sided bounds on the optimal value function (see e.g. [83, 108, 127]).

It is also possible to find solutions to SSPP:s using algorithms based on the AO* search algorithm [87]. The solution is then not a policy but a conditional plan of actions. The AO* algorithm is a heuristic search algorithm for acyclic

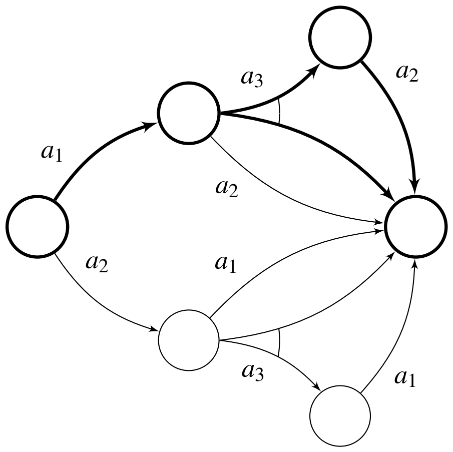

Figure 2.8: A small example of an AND/OR graph with a possible solution highlighted in bold. Hyperedges are drawn with directed edges joined by a small arc.

AND/OR graphs. An AND/OR graph for an SSPP is a directed hypergraph where the nodes correspond to decision points, each labeled with a belief state, and the edges are hyperedges, each labeled with an action directed to one other node for each possible outcome of that action. The root node in this graph is labeled with the initial belief state. A *solution* to an AND/OR graph $\mathcal{G}$ is a subgraph $\mathcal{G}'$ such that it is rooted in the initial belief state and every non-leaf node in $\mathcal{G}$ that is reachable in $\mathcal{G}'$ has exactly one outgoing edge, i.e. a single decision is made at every decision point. This subgraph represents a conditional plan. The cost of a solution $\mathcal{G}'$ rooted in a belief state $b$ is the expected cost of reaching a leaf node from $b$:

$$V(b) = \bar{c}(b, \pi(b)) + \sum_{o \in \mathcal{O}} \bar{p}(b, \pi(b), o) V(b_{\pi(b),o}) \tag{2.14}$$

where $\pi(b)$ is the action corresponding to the outgoing edge of the node labeled with $b$ in $\mathcal{G}'$.

Figure 2.8 shows an example of an AND/OR graph for Example 2.3 rooted in a belief state $b_0$ where $b_0(s_1) = 0.75$ and $b_0(s_2) = 0.25$ (same as Example 2.2). The optimal solution is highlighted in bold. This AND/OR graph corresponds to the decision tree in shown Figure 2.6.

The AO* algorithm finds the optimal solution by iteratively expanding a search graph that is a subgraph of the full AND/OR graph. At all times, the algorithm records the lowest-cost solution for this subgraph. For each node of the search graph, a value function is used to represent an estimate of the solution cost from that node to the goal. The search graph is expanded by selecting a leaf node in the current solution that is not a leaf node in the full AND/OR graph. Whenever a node has been expanded, changes to the values of other nodes and the current best solution are propagated back to the root by performing Bellman updates on all ancestor nodes. A heuristic function is used to assign costs to unexpanded leaf nodes. The algorithm stops when no more nodes can be expanded. Then the solution costs of all nodes in the search graph are minimal. Like the A* algorithm [59], from which the AO* algorithm borrows its name, the final solution will be optimal if the heuristic is an admissible heuristic that does not overestimate the optimal expected cost.

The basic AO* algorithm cannot handle problems with cycles in the AND/OR graph, but variants such as the LAO* algorithm [58] can handle more general AND/OR graphs with cycles.

See e.g. [4, 29] for applications of the AO* algorithm for diagnosis and troubleshooting.

### 2.2.3 Case-Based Reasoning

The Case-Based Reasoning approach to making decisions for diagnosis and troubleshooting is to take decisions based on which decisions have been made previously when similar observations have been seen (see e.g. [45, 77] for a general overview). When Case-Based Reasoning is applied to troubleshooting, situational information regarding which observations were seen and which repair action resolved the problem is stored in a *case library*, whenever the troubleshooting of the system has been successfully completed. Then when another system needs troubleshooting, the current observations are matched with similar cases in the case library. If the same repair action has resolved the problem for all or most of the matching cases, this action will be recommended to the user. Otherwise observing actions can be recommended in order to discriminate between cases in the case library. The case library can initially be filled with cases from manual troubleshooting sessions and then, as more cases are successfully solved using the Case-Based Reasoning system, the quality of the recommended decisions improves over time [43].

There are several applications of Case-Based Reasoning for troubleshooting in the literature (see e.g. [16, 30, 43, 52]). However, these applications do not explicitly consider the problem of making decisions that minimize the expected cost of repair. Also, being purely data-driven, they require large amounts of data from comparable systems to function effectively.

# Chapter 3

# Contributions

This chapter gives a detailed overview of the troubleshooting framework that has been developed. The research has led to contributions within four related research areas: the overall troubleshooting framework, diagnosis, planning, and applications. Table 3.1 shows, for each of the publications authored or co-authored by the author of this thesis, in which of the research areas related to the problem formulation there has been a contribution. The following sections will discuss the contributions made in all of these publications in some detail. The publications [6, 8, 9, 11, 12] constitute the most significant contributions and these are appended as Papers A–E in the second part of the thesis.

## 3.1 Troubleshooting Framework

This thesis work has led to the development of a framework for computer assisted off-board troubleshooting that can be applied to vehicles in the workshop and remotely to vehicles on the road. For the case when the vehicle is in the workshop only, the framework is presented in part in [1, 4, 8] and in full in [7]. An extension of the framework so that remote troubleshooting can be supported is presented in [11, 12]. Prototype implementations of the framework have been created for a hydraulic braking system [7, 8], an engine temperature control system, [11], and a fuel injection system [12].

| | Title | Framework | Diagnosis | Planning | Application |
|---|---|---|---|---|---|
| [1] | Troubleshooting when Action Costs are Dependent with Application to a Truck Engine | | | x | x |
| [2] | A Heuristic for Near-Optimal Troubleshooting Using AO* | | | x | x |
| [3] | Modeling and Troubleshooting with Interventions Applied to an Auxiliary Truck Braking System | | x | | x |
| [4] | Anytime Near-Optimal Troubleshooting Applied to a Auxiliary Truck Braking System | | | x | x |
| [5] | Planning as Heuristic Search for Incremental Fault Diagnosis and Repair | | | x | x |
| [6] | Iterative Bounding LAO* | | | x | |
| [7] | Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis | x | x | x | x |
| [8] | Modeling and Inference for Troubleshooting with Interventions Applied to a Heavy Truck Auxiliary Braking System | x | x | | x |
| [9] | Exploiting Fully Observable and Deterministic Structures in Goal POMDPs | | | x | |
| [10] | Improving the Maintenance Planning of Heavy Trucks using Constraint Programming | | | x | x |
| [11] | Guided Integrated Remote and Workshop Troubleshooting of Heavy Trucks | | | x | x |
| [12] | A Modeling Framework for Troubleshooting Automotive Systems | x | x | | x |

Table 3.1: Research areas in which contributions have been made for each publication authored or co-authored by the author of this thesis. The above publications are also listed in the same order first in the bibliographic references section of this thesis.

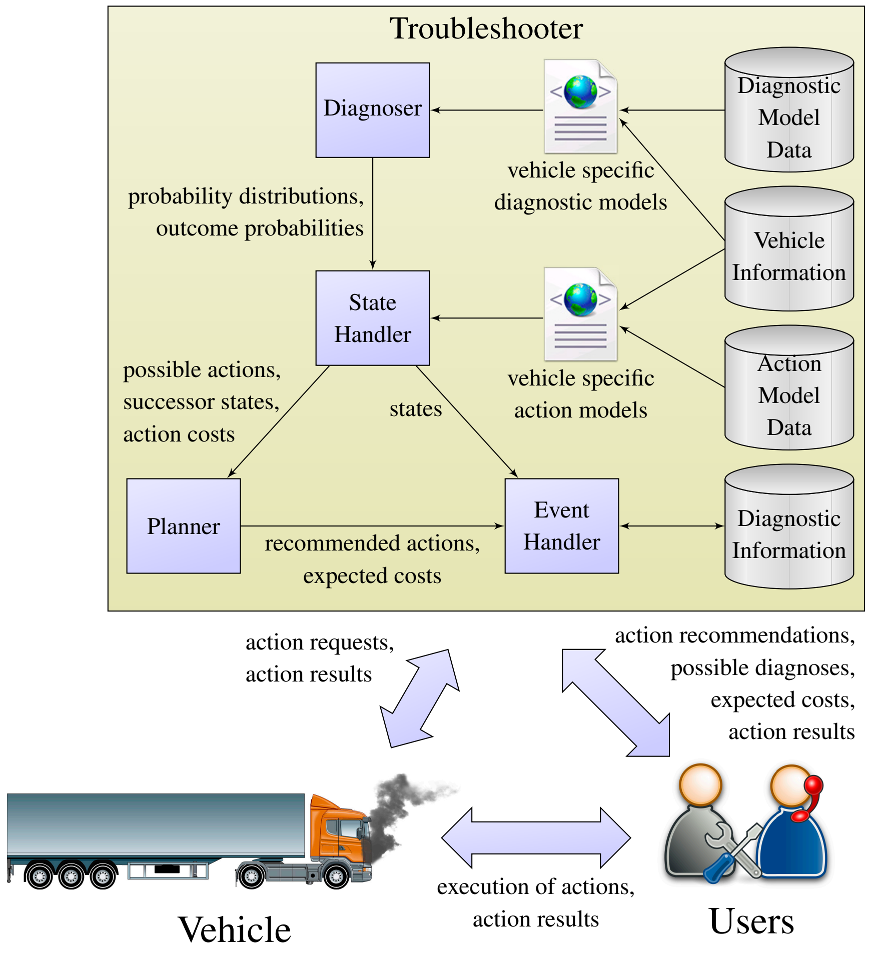
Troubleshooter
Diagnoser
vehicle specific diagnostic models
Diagnostic Model Data
probability distributions, outcome probabilities
Vehicle Information
State Handler
vehicle specific action models
Action Model Data
possible actions, successor states, action costs
states
Planner
recommended actions, expected costs
Event Handler
Diagnostic Information
action requests, action results
action recommendations, possible diagnoses, expected costs, action results
execution of actions, action results
Vehicle
Users

Figure 3.1: Overview of the troubleshooting framework.

Figure 3.1 shows an overview of the troubleshooting framework. The users of the troubleshooter can either be in the workshop together with the vehicle (e.g. mechanics) or interact with the vehicle remotely (e.g. helpdesk personnel). They use the troubleshooter to get recommendations of actions, estimates of the expected cost, and explanations of possible diagnoses. When a user has executed an action on the vehicle, the action result is fed back to the troubleshooter afterward by the user. It is also possible that the troubleshooter directly executes an action on the vehicle (e.g. remote tests) and retrieves the action results by itself.

The *troubleshooter* consists of four modules, the *event handler*, the *state handler*, the *diagnoser*, and the *planner*:

- The event handler is the module with which users interact. It retrieves and stores dynamic diagnostic information about the vehicle in a database for diagnostic information in the form of *events*. The events describe changes related to the vehicle, e.g., that a component is repaired, an observation is made, and the vehicle is operated. The event handler uses the state handler to obtain a state description of the vehicle by providing it with a previous state and a sequence of events that have occurred since that state was computed. The state includes information about the probabilities of possible faults given the events, which can be presented to the user. To obtain action recommendations and estimates of the expected cost, the event handler uses the planner by providing it with an initial state to start planning from.

- The state handler is responsible for computing the state of the vehicle given a previous state and a sequence of events. It also provides the planner with information about which actions are possible given a state, what these actions cost, and which other states each action can reach with which probability. For this it uses an *action model* that contains information of which actions there are and their costs, preconditions, and effects. The action model is compiled to be specific for the current vehicle of the troubleshooting session using vehicle information stored in a separate database. When computing new states and the probabilities of reaching them, the state handler uses the diagnoser to compute fault and outcome probabilities.

- The diagnoser is responsible for providing the state handler with fault and outcome probabilities. Depending on the implementation of the diagnoser, these probabilities are computed given a set of previous observations [11] or a previous probability distribution and a new event [1, 4, 7, 8, 12]. The diagnoser uses a *diagnostic model* that describes probabilistic dependencies between component faults and observations. As with the action model, the diagnostic model is compiled to be specific for the current vehicle of the troubleshooting session using the vehicle information.

- The planner is responsible for providing the event handler with action recommendations and estimates of the expected cost. It uses the state handler to get information of which actions are possible for a given state, what their costs are, and which states are reached with which probability.

The framework is designed so that different implementations of the modules can with some limitations be used interchangeably. The information that the state handler provides to the planner is general enough so that different planning algorithms can be plugged in and tested with different settings and heuristics. For example, information of primitive actions, such as removing bolts and screws in order to reach a component to replace or perform tests on, can be hidden away from the view of the planner inside the state handler and be replaced with macro actions consisting of sequences of primitive actions. Then, the same implementation of the planner that operates on the primitive actions can be used with the macro actions (see e.g. [1, 4, 9]).

Different diagnosers can be used as long as they can provide the needed fault and outcome probabilities. For example in [1, 4, 5, 7, 11] the diagnosers are implemented using static Bayesian networks, and in [3, 7, 8, 12] non-stationary dynamic Bayesian networks are used.

## 3.2 Diagnosis

Several contributions have been made for this thesis on how to solve the diagnosis problem and how to implement a diagnoser. In [3] and [8] (Paper B), a novel method for making diagnosis using non-stationary Bayesian networks (nsDBN) is presented. In [7], further improvements and generalizations of this method are made where the nsDBN is converted back into a single static Bayesian network. In [12] (Paper E), a novel diagnostic framework using dynamic Bayesian networks is presented. This framework is better suited for integrated remote and workshop troubleshooting where the troubleshooting sessions span over longer periods of time. For this framework, it is described how model parameters can be learned from statistical data and several different inference methods are evaluated.

### 3.2.1 Troubleshooting with Interventions using nsDBN:s

If it is possible to actively change the health state of components by either making repairs or operating the vehicle, then it can be problematic to model the troubleshooting process using an ordinary stationary Bayesian network like the one shown in Figure 2.5. These types of actions change the value of random variables in the network in a non-random way. Such a change is called an *intervention* and it influences how inference can be made in the network [94]. In other approaches [62, 75], interventions due to repairs are handled by enforcing a rule that requires the user to verify whether the last repair action solved the problem or not. If the repair did not solve the problem, then we know that there was no fault on the repaired component and we can treat the repair as an ordinary observation of the health state of that component being non-faulty. If the repair did solve the problem we know that the system has no faults and further inference is not needed. However, when troubleshooting trucks the cost of verifying whether a repair has solved the problem can be large in comparison with the cost of the repairs themselves. Enforcing this rule of mandatory problem verifications can make troubleshooting unnecessarily expensive and therefore novel methods for modeling the troubleshooting process are needed.

In [8] we propose to model the troubleshooting process with an event-driven nsDBN. In distinction to an ordinary dynamic Bayesian network, each time slice of an nsDBN can have a different structure and different conditional probability distributions [102]. In our case these differences come from repairs and the operations of the vehicle which generate events that are interventions. For each new event, a new time slice is added to the nsDBN whose structure and parameters are dependent on the event. Straight-forward inference in such a network using ordinary methods is intractable because the size of the network grows over time. The proposed solution is to use a static Bayesian network and manipulate the structure and parameters of this network using a set of rules for each new event. The manipulation according to these rules ensures that inference in this static network is equivalent to inference in the full nsDBN for the types of queries needed in troubleshooting.

Figure 3.2 shows an example from the paper of how such a static network is manipulated. In the model, random variables are classified as either *persistent* or *non-persistent* and probabilistic dependencies are classified as either *instant* or *non-instant*. A random variable is persistent when its value is dependent on the random variable corresponding to itself in a previous time slice (gray nodes). Otherwise it is non-persistent (white nodes). An instant dependency between variables (solid line) is present between the variables in every time-slice. A non-instant dependency between variables (dashed line) is present only after operation events. After other events, the child variable is dependent on the random variable corresponding to itself in the time slice after the most recent operation event. A limitation of the proposed method is that the equivalence between the static network and the nsDBN only holds if there is at least one operation event between any two repair events.

An extension to the method presented above is proposed in [7]. This method does not have the same limitation and therefore the events can occur in arbitrary order. The solution is to keep two copies of the persistent variables, one representing their values at the time before the last operation event and another representing their values at the current time. Figure 3.3 shows an example of the static Bayesian network representation of an nsDBN using this method. Every non-instant edge from a persistent variable appears in the static Bayesian network as an edge from the copy of the persistent variable at the time before the operation event and every instant edge from a persistent variable appears from its copy at the current time. Theoretical results show that given a probability distribution over all the persistent variables, inference in the static Bayesian network is equivalent to inference in the nsDBN for the relevant query types. An apparent weakness with this method is the requirement to have to know the probability distribution over the persistent variables both at the current time and at the time of the last operation event. However, it is shown that it is sufficient to only keep a probability distribution over the persistent variables at the time of the last operation event and the set of repair events

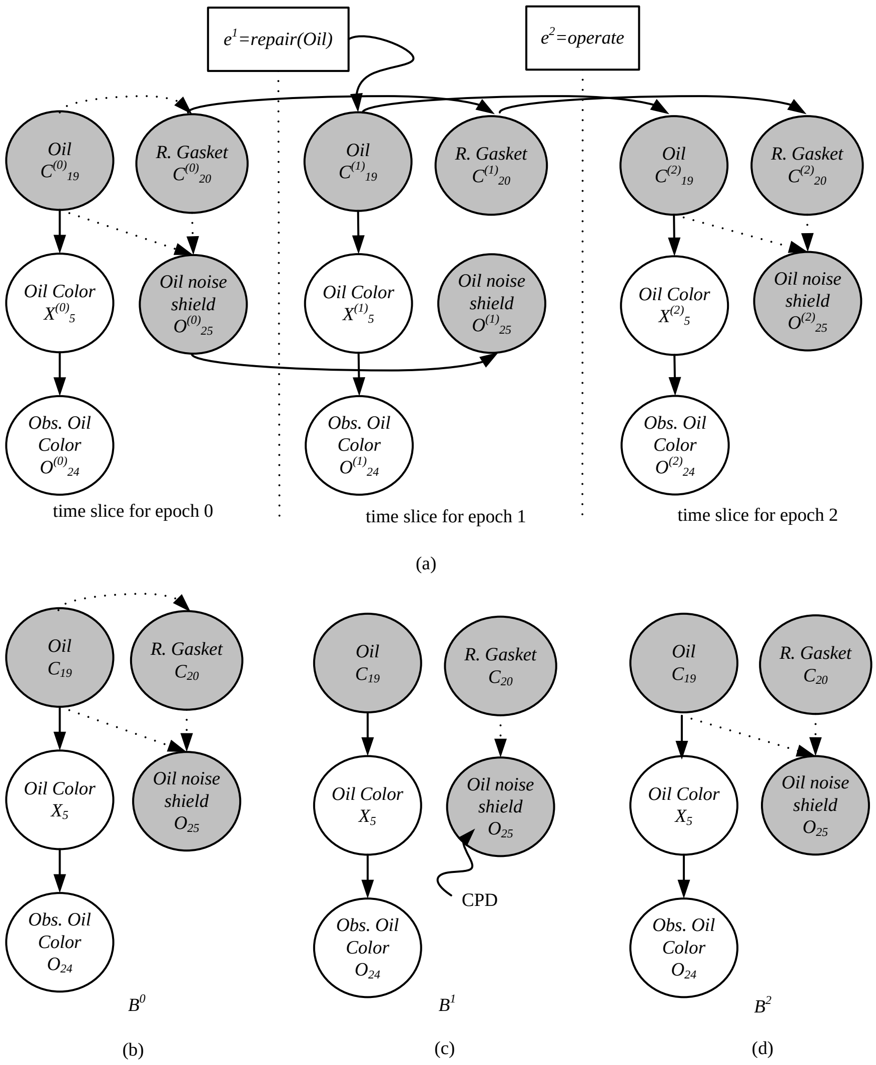

Figure 3.2: An nsDBN modeling a small example system described in [8] subject to a troubleshooting sequence (top), and the corresponding static Bayesian networks (bottom). Persistent variables are highlighted in gray and non-instant edges are dashed. When a repair is made on $C_{19}$ the conditional probability distribution on the persistent descendant $O_{25}$ is altered to consider the value of $C_{19}$ before the repair. When an operation event occurs all non-instant dependencies are reinserted.

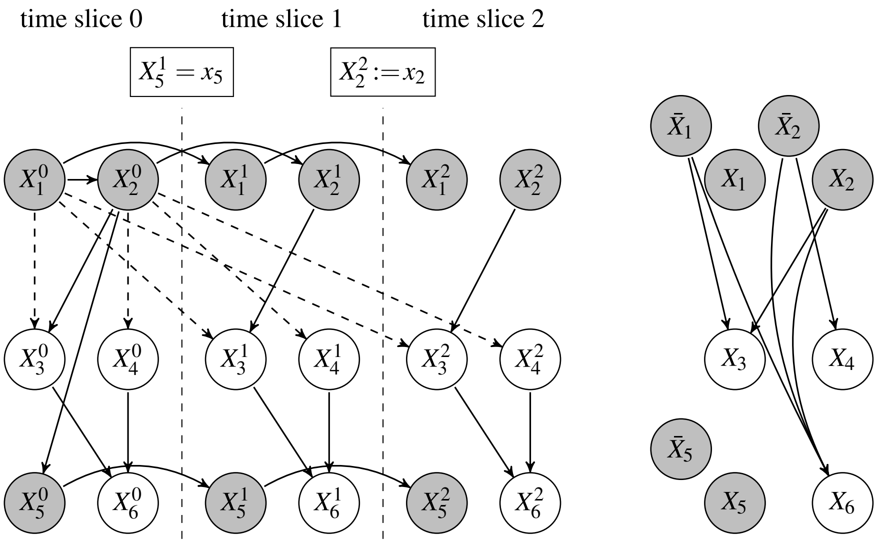

Figure 3.3: The first three time slices of the nsDBN in Example 2.5 in [7] are shown to the left and the corresponding static Bayesian network is shown to the right. Persistent variables are highlighted in gray and non-instant edges are dashed.

that have occurred since that time. Then, the probabilities of the persistent variables at the current time can be derived from this probability distribution and the known repair events. This method is shown to be feasible for making diagnostic inference on a model of a subsystem of a truck: an auxiliary braking system with 20 components that can have faults.

### 3.2.2 Diagnostic Framework for Integrated Remote and Workshop Troubleshooting

In the previous methods presented in Section 3.2.1 it is assumed that no new faults are introduced during operation events. This is because the frameworks are intended to be used when the vehicle is in the workshop during the entire troubleshooting session. In the workshop, vehicles are typically operated for time periods in the order of minutes. This can be compared to the mean time between failures for components which is in the order of years. In [12] a novel diagnostic framework is proposed that is intended to be used when the planning of troubleshooting actions that can be performed remotely while the vehicle is on the road is integrated with the planning of the troubleshooting actions that are carried out in the workshop. In this case, the time from the moment when the first observation of a problem occurs and remote troubleshooting can be initiated until the time the vehicle leaves the workshop and the troubleshooting session ends can be several weeks. Over time periods of this length it is not feasible to assume that the persistent variables remain unchanged during operation. While awaiting an appropriate time for making a workshop visit, new faults can occur and the behavior of faults can change so that previously observed symptoms disappear and new symptoms appear.

In the proposed solution, the diagnostic model of the vehicle is modeled with different types of random variables representing *components*, *faults*, *symptoms*, *faults causing symptoms*, and *observations*. Figure 3.4 shows the topology of dependencies between these variables for a small example with a single component: a rail pressure sensor that can have three different types of faults. The model is modular in the sense that several component-level submodels can be combined into a complete model to match the configuration of a particular vehicle. Each of these submodels can be created and parameterized independently.

The presence of a fault $i$ at time $t$ is modeled with a *fault variable* $F_i^t$ which is a random variable that can either have the value *present* (P) or the value *not present* (NP). The process of faults occurring during operation is modeled using a homogenous Poisson process. When this model is used, faults are considered to be equally probable to occur at any time during operation and there exist exact and efficient methods for computing probabilities and learning parameters from statistical data. The probability of having a fault at time $t_2$ is only dependent on the probability of having a fault at a previous time point $t_1$ and the number of kilometers traveled since then:

$$\Pr(F_i^{t_2} = \mathrm{P}) = \Pr(F_i^{t_1} = \mathrm{P}) + \left(1 - e^{-\lambda_{fail,i}(m(t_2)-m(t_1))}\right)\Pr(F_i^{t_1} = \mathrm{NP})$$

where $\lambda_{fail,i}$ is a parameter that describes the mean time (in operated kilometers) between failures for fault $i$ and $m(t)$ is the mileage at time $t$.

A novel model is used for modeling how symptoms are caused by faults. This model is similar to the Noisy-Or distribution [63] where a symptom can be present only if there is a fault present that causes the symptom. The Noisy-Or model is extended with a model describing how symptoms can appear and disappear intermittently as the vehicle is operated. The causing of a symptom $j$ by a fault $i$ at time $t$ (a symptom-cause) is modeled with a *symptom-cause variable* $S_{i,j}^t$ which is a random variable that can either have the value *active* (A) or *not active* (NA). A symptom (modeled with a *symptom variable*) is present if any of its symptom-causes are present. Unless its fault is present, a symptom-cause cannot be active, but if the fault is present then the symptom-cause does not necessarily have to be active. A parameter $p_{i,j}$ models the probability that a symptom-cause becomes active when the fault $i$ occurs. If the fault should already be present and the vehicle is operated for some time, then the symptom cause may transition from active to non-active and vice versa. The frequency of these transitions is controlled with a

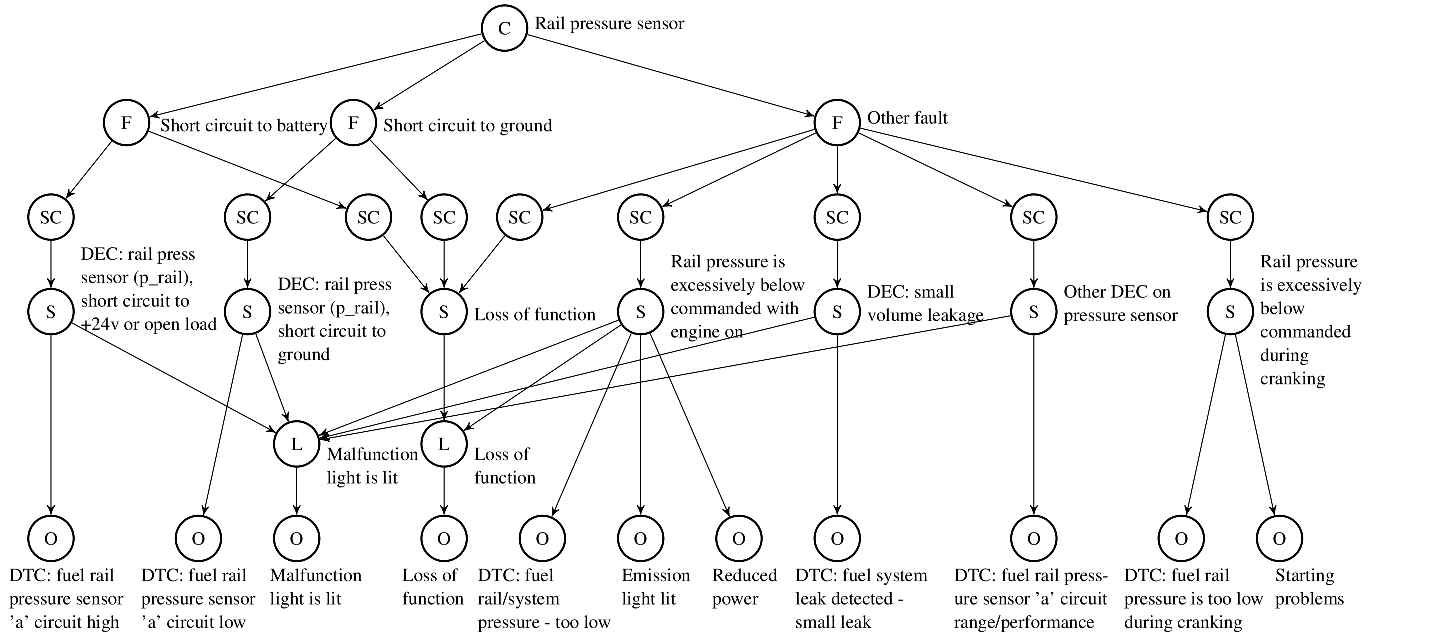
C
Rail pressure sensor
F
Short circuit to battery
F
Short circuit to ground
F
Other fault
SC
S
L
O
DEC: rail press sensor (p_rail), short circuit to +24v or open load
DEC: rail press sensor (p_rail), short circuit to ground
Loss of function
Rail pressure is excessively below commanded with engine on
DEC: small volume leakage
Other DEC on pressure sensor
Rail pressure is excessively below commanded during cranking
Malfunction light is lit
Loss of function
DTC: fuel rail pressure sensor 'a' circuit high
DTC: fuel rail pressure sensor 'a' circuit low
Malfunction light is lit
Loss of function
DTC: fuel rail/system pressure - too low
Emission light lit
Reduced power
DTC: fuel system leak detected - small leak
DTC: fuel rail pressure sensor 'a' circuit range/performance
DTC: fuel rail pressure is too low during cranking
Starting problems

Figure 3.4: The model topology for an example of a fuel rail pressure sensor described in [12]. Nodes labeled with *C* are component variables, *F* are fault variables, *SC* are symptom cause variables, *S* are symptom variables, *L* are logical variables (help variables for expressing logical dependencies between observations and symptoms), *O* are observation variables. A Diagnostic Error Code (DEC) is the result of an on-board test that can trigger a DTC.

parameter $\lambda_{i,j}$ called the *symptom-cause inertia*. If the vehicle is operated between times $t_1$ and $t_2$, then with probability $e^{-\lambda_{i,j}(m(t_2)-m(t_1))}$ the symptom cause will remain the same, otherwise it will be active with probability $p_{i,j}$. The distribution of $S^t_{i,j}$ depends on two parameters, $p_{i,j}$ and $\lambda_{i,j}$. Assume that the vehicle is operated from time $t_1$ to time $t_2$. The probability that the symptom cause is present at time $t_2$ is:

$$
\begin{aligned}
\Pr(S^{t_2}_{i,j} = \mathrm{A}) = \Pr(F^{t_1}_i &= \mathrm{NP})p_{i,j}\big(1 - e^{-\lambda_{fail,i}(m(t_2)-m(t_1))}\big) \\
&+ \Pr(F^{t_1}_i = \mathrm{P})\Bigg(\Pr(S^{t_1}_{i,j} = \mathrm{NA})p_{i,j}\big(1 - e^{-\lambda_{i,j}(m(t_2)-m(t_1))}\big) \\
&+ \Pr(S^{t_1}_{i,j} = \mathrm{A})\Big(e^{-\lambda_{i,j}(m(t_2)-m(t_1))} + p_{i,j}\big(1 - e^{-\lambda_{i,j}(m(t_2)-m(t_1))}\big)\Big)\Bigg).
\end{aligned}
$$

These symptoms represent what is actually true on the system. However, an external observer may be unable to correctly observe this. What is observable by the external observer is modeled with binary *observation variables*. If an observation should depend on several symptoms, a logical function is used to map every possible combination of the values of these symptoms into a binary response. The value of this function is represented with a *logical variable*. Typically there is a one-to-one correspondence between the value of a symptom variable or a logical variable and its corresponding observation variable. However, if it is possible that the external observer perceives the symptoms erroneously, e.g. due to measurement noise, the probabilities of having a false positive or a false negative result of observations are modeled with Bernoulli distributions.

When an event occurs, a new set of variables is representing the vehicle after this event. The CPT:s of these variables depend on the occurred event which means that this model is also an nsDBN. As previously mentioned, making inference directly in an nsDBN is intractable. Instead of finding a static Bayesian network equivalent of the nsDBN, we explore several different approximate inference methods based on the Boyen-Koller inference algorithm for dynamic Bayesian networks [26]. For this, a new model for approximately representing the probability distribution over the persistent variables (the fault and symptom-cause variables) is proposed that combines a single fault model with a multiple fault model. Also, a new inference algorithm is presented that is capable of making inference in linear time when the network has a certain structure. This algorithm is inspired by the Quickscore algorithm for two-layer networks with only Noisy-Or CPT:s [61]. The different inference methods using different representations of the probability distributions are evaluated both empirically and theoretically with regard to correctness and efficiency. It is shown that the approximate inference using the new inference algorithm together with the new method for representing the probability distributions is feasible for models with thousands of possible faults.

The framework also includes methods for learning parameters from data based on a combination of conjugate priors and maximum à posteriori estimates. These methods are also evaluated both empirically and theoretically and it is shown that it is possible to learn model parameters from reasonable amounts of statistical data in the form of warranty reports.

## 3.3 Planning

Several contributions have been made on how to improve the planner. In [6] (Paper A), a novel algorithm is proposed for problems formulated as Stochastic Shortest Path Problems (SSPP:s). This algorithm, Iterative Bounding LAO*, is a general algorithm that has properties that make it more suitable for planning troubleshooting problems than other comparable algorithms. In [1, 4], a method is presented for separating actions that assemble and disassemble parts of the vehicle from those that make observations and repair faults. Using this method, the planning problem can be solved more efficiently using standard planning algorithms. In [9] (Paper C), this method is generalized so that it can be applied on any problem formulated as a goal-POMDP. Novel heuristics that can be used by the different planning algorithms are described in [2, 5, 7]. In [11] (Paper D) an extension to the troubleshooting framework is presented where remote and workshop troubleshooting is integrated. The planner used in this framework is responsible for planning actions from the instant a fault is detected on the road until the vehicle leaves the workshop where both repair costs and vehicle downtime costs are considered. A novel planning algorithm is proposed that achieves this.

### 3.3.1 Iterative Bounding LAO*

Iterative Bounding LAO* (IBLAO*) finds near-optimal solutions to problems formulated as SSPP:s. It is based on the algorithm LAO* [58] (see Section 2.2.2) and uses two search heuristics to guide the search. Just as LAO*, it searches the state space from an initial state until a conditional plan of actions is found that achieves the goal for every possible action outcome. During search it keeps track of estimates of the minimal expected cost to reach the goal from each searched state. In distinction to LAO*, it uses two estimates for the minimal expected cost for each searched state $s$: a lower bound estimate $f_l(s)$ and an upper bound estimate $f_u(s)$. During search the distance between the estimated bounds is iteratively narrowed over time towards the minimal expected cost.

A lower bound policy $\pi_l$ and an upper bound policy $\pi_u$ are defined such that for each state $s$, $\pi_l(s)$ is the action that minimizes the lower bound and $\pi_u(s)$ is the action that minimizes the upper bound. In each iteration of the algorithm, the values of the estimates and the policies are updated on selected states by applying the backup operations, (2.12) and (2.13), on these. The states that are backed up are selected from the fringe nodes of the solution graph of the lower bound policy. For previously unsearched states, the bounds must be initialized using heuristics $h_l$ and $h_u$ such that for all states $s$,

$$f_l(s) = h_l(s) \leq V^*(s) \leq h_u(s) = f_u(s)$$

where $V^*(s)$ is the minimal expected cost to reach the goal from state $s$. Then after each backup, the new values of the estimates, $f'_l(s)$ and $f'_u(s)$, will be closer or equally close to the minimal expected cost than the previous values, $f_l(s)$ and $f_u(s)$:

$$f_l(s) \leq f'_l(s) \leq V^*(s) \leq f'_u(s) \leq f_u(s).$$

Also, it is shown that the minimal expected cost of using the upper bound policy is smaller or equal than the upper bound if the upper bound heuristic is uniformly improvable, i.e. if $f_u(s) = h_u(s)$ for all states $s$ then after backing up $s$, $f'_u(s) \leq h_u(s)$. In practice, this is achieved if the upper bound heuristic is derived from a real policy that solves the problem suboptimally.

The novelty of IBLAO* is the mechanism for expanding the search graph so that the total number of needed expansions is kept small. States to expand the search graph with are selected from those fringe nodes that have a high probability of being reached with the lower bound policy and where the relative difference between the upper and lower bound is large. During search, the algorithm can output solutions to the SSPP with proven bounds at any time. The user can monitor the bounds at run time and use these to decide when a solution with sufficient quality is found. It is also shown how IBLAO* can be used together with weighted heuristics for faster initial convergence of the bounds.

Two-sided bounds have been used in other algorithms for SSPP:s that are based on the RTDP algorithm [83, 108, 127]. In empirical tests on benchmark problems, IBLAO* requires significantly fewer state expansions at the cost of a higher overhead when propagating bounds because it performs more state backups. For troubleshooting problems it is computationally intensive to expand states because this requires inference in a Bayesian network. Then the lower number of state expansions for IBLAO* is a clear advantage.

### 3.3.2 Assembly and Disassembly Actions

When troubleshooting a vehicle in a workshop, many actions are performed that do not directly serve to make an observation or a repair, e.g. actions that disassemble components in order to reach parts of the vehicle where a component can be repaired and actions that reassemble the vehicle after the troubleshooting is completed. The cost of these actions needs to be considered during planning because it can stand for a significant part of the total troubleshooting cost.

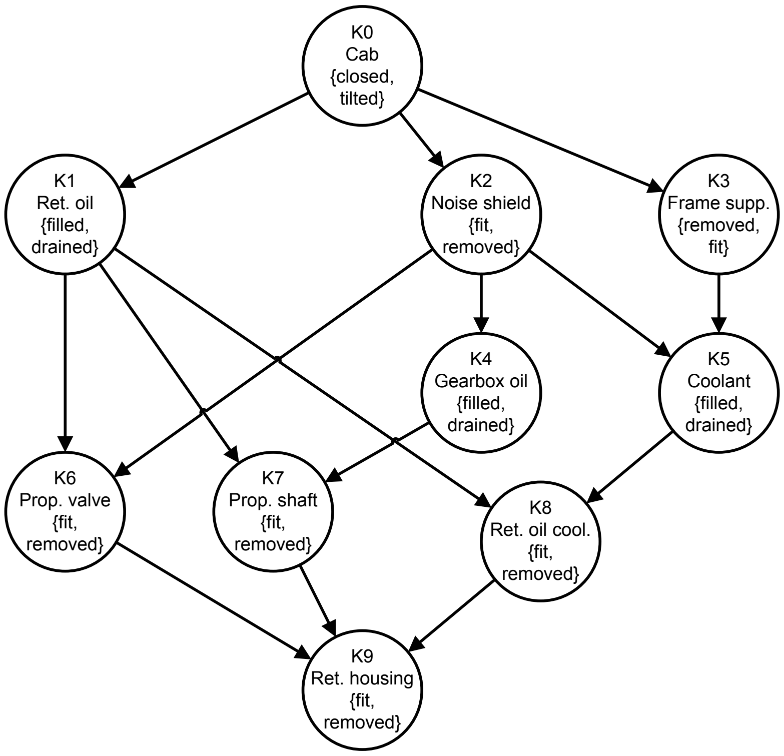

Figure 3.5: The assembly graph for a hydraulic braking system [5].

In [4] it is proposed to separate the planning of observing and repairing actions from the planning of the actions that assemble or disassemble decomposable parts of the vehicle. These decomposable parts are represented with binary variables called *assembly variables* that can either be in the state *assembled* or *disassembled*. For every assembly variable there is an *assembly action* that makes the variable assembled and a *disassembly action* that makes the variable disassembled. Each action that can be performed on the vehicle has a precondition that describes which assembly variables must be assembled and which must be disassembled for the action to be executable.

The preconditions of the assembly and disassembly actions are modeled with an *assembly graph* which is a directed acyclic graph that describes in which order decomposable parts of the vehicle must be disassembled. Figure 3.5 shows an example of such a graph for a hydraulic braking system. In order to assemble an assembly variable, all children of this variable in the assembly graph must be assembled, and, vice versa, in order to disassemble an assembly variable, all parents of this variable must be disassembled. The sequence of assembly and disassembly actions needed to fulfill the precondition of another action can be computed quickly through a topological search of this graph (see Algorithm 4 in [7]). Then a *macro action* is created by joining this sequence of actions with the intended repair or observing action.

During planning, the planning algorithm only needs to consider these macro actions because the assembly and disassembly actions are hidden away in the cost function. The benefit with this approach is that a goal state can be reached in much fewer steps which thereby potentially reduces the size of the search graph. Empirical tests on a troubleshooting model of an auxiliary braking system shows that this approach can speed planning up with several orders of magnitude (see Table 5.7 in [7]).

### 3.3.3 Exploiting Deterministic Structures

In [9], a generalization of the above method of abstracting away assembly and disassembly actions is proposed. This is a general theory that can be applied to any problem modeled as a factored goal-POMDP. It is based on the principle that actions that deterministically affect only variables that are fully observable can be abstracted away just as we previously abstracted away the assembly and disassembly actions. The input to the method is a factored goal-POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, t, p, c, 1 \rangle$ with initial state $s_0$.

First the goal-POMDP model is preprocessed in a three step process identifying the fully observable variables and the actions that can be abstracted away. In the first step, the model is scanned so that fully observable variables can be identified. A variable is found to be fully observable if its value is known in the initial state and every action either deterministically manipulates it or makes an observation that uniquely identifies its value. The variables that are not fully observable are said to be partially observable. Each state $s \in \mathcal{S}$ can then be separated into a fully observable part $fo(s)$ indicating the values of the fully observable variables, and a partially observable part $po(s)$ indicating the values of the partially observable variables.

In the second step, the action set $\mathcal{A}$ is transformed so that all actions can be assigned valid *FO-preconditions*. A FO-precondition of an action is a boolean function of the fully observable part of the state. To be a *valid* FO-precondition, it must hold that for all states where the FO-precondition evaluates to true, the effect of the action must be *similar* with regard to its cost and its effect on the partially observable part of the state. An action $a$ is defined to have similar effect in the states $s_i, s_j$ if:

$$c(a, s_i) = 0 \text{ iff } c(a, s_j) = 0$$

and for all partially observable variables $V$:

$$\Pr(V = v|s_i, a) \begin{cases} = 0 & \text{if } \Pr(V = v|s_j, a) = 0, \\ \in (0,1) & \text{if } \Pr(V = v|s_j, a) \in (0,1), \\ = 1 & \text{if } \Pr(V = v|s_j, a) = 1. \end{cases}$$

where $\Pr(V = v|s,a)$ is the probability that executing action $a$ in state $s$ results in a state where $V$ has the value $v$. In general, not all actions in the original action set have valid FO-preconditions. However, an action for which valid FO-preconditions cannot be found can be partitioned into multiple actions with valid FO-preconditions for non-overlapping subsets of the state space.

In the third step, the actions are classified as either *support actions* or *relevant actions* where a support action is an action that only deterministically manipulates fully observable variables, and a relevant action is an action that can gain information of the belief state or manipulate a state variable in a way that a support action cannot. When comparing to the previous approach, the assembly and disassembly actions would be classified as support actions and the observing and repairing actions would be classified as relevant actions.

When a planning algorithm needs to know which actions are applicable in a given belief state, a macro action is created for each relevant action by computing the sequences of support actions needed to satisfy the FO-precondition of that action. This can be done using any shortest path algorithm such as the A* algorithm [59].

Theoretical results show that the method is *complete*, i.e. if there exists a solution to the Goal-POMDP with a finite cost, then such a solution can be found in the transformed model. Because of the way that valid FO-preconditions are defined, it does not matter in which fully observable state that a relevant action is performed as long as the FO-precondition is true.

A drawback with the method is that when the model has many actions that behave differently in different parts of the state space, the number of relevant actions can increase significantly because of the splitting of actions. However when the actions are such that they are only applicable in very small parts of the state space, the method can instead speed up planning significantly. This is the case for problems similar to the troubleshooting problem and empirical results support this claim.

### 3.3.4 Heuristics

Many planning algorithms that can be used for the troubleshooting problem benefit from search heuristics. In [2] a heuristic is proposed where parameters are learned using training data from simpler problem instances. Some planning algorithms, such as the IBLAO* algorithm, require that the heuristics are either lower or upper bounds of the minimal expected cost. In [7] several such heuristics are proposed that can be used in various settings of the troubleshooting problem.

### Lower Bound Heuristics

When the SSPP is a belief-MDP, it is possible to create a lower bound heuristic by solving a simplified problem where the true state is fully observable [15]. The heuristic estimate of the expected cost in a belief state can then be obtained by weighting the probability of being in a particular state with the cost of solving the problem when the system is known to be in that state. When applied to the troubleshooting problem this is equivalent to assuming that we can always determine which fault is present with zero cost. For each fault that is possible, the optimal plan for repairing the now known fault is deterministic and can easily be computed. In [7] this is called the *full observability* heuristic $h_{fo}$:

$$h_{fo}(b) = \sum_{d} \Pr(d|b) c_{rep}(d,b)$$

where $\Pr(d|b)$ is the probability of having the diagnosis $d$ given belief state $b$ and $c_{rep}(d,b)$ is the cost of repairing all faults in $d$ given the fully observable part of $b$.

A drawback of the full observability heuristic is that it does not consider the costs of the observing actions that might be needed to solve the problem. When only observing actions are considered, it is common to use the *entropy* of the belief state as a measure of the expected cost to solve the problem (see Section 2.2.1). We will call this the *entropy heuristic* $h_{ent}$:

$$h_{ent}(b) = H(b) c_H$$

where $H(b) = \sum_s b(s) \log_2(b(s))$ is the entropy of belief state $b$ and $c_H$ is a constant. If $c_H$ is the smallest cost of reducing the entropy by 1 using any sequence of observing actions, then $h_{ent}$ is a lower bound.

Because the entropy heuristic only considers the cost of observing actions and the full observability heuristic only considers the cost of repair actions it is natural to form another heuristic by summing these two as in [132]. In [2], it is proposed to use such a combined heuristic where the value of $c_H$ is learned from training data collected from previously solved problem instances for better planning performance. However, the heuristic obtained by straight-forwardly adding $h_{fo}$ to $h_{ent}$ is not a lower bound because it does not consider that repair actions also can reduce the entropy of the belief state. In [7], a new heuristic that combines $h_{fo}$ and $h_{ent}$ so that it becomes a lower bound is proposed. This is the *combined heuristic* $h_{comb}$:

$$h_{comb}(b) = h_{fo}(b) + \max\left(0, H(b) - \sum_{d} \Pr(d|b) H_{rep}(d)\right) c_H$$

where $H_{rep}(d)$ is the maximal entropy that can be removed by repairing all faults in the diagnosis $d$. It is shown both theoretically and empirically that this heuristic is a lower bound and that it is stronger than both $h_{ent}$ and $h_{fo}$ individually. The above formulation of the combined heuristic assumes that all repair actions are perfect, i.e. attempted repairs always succeed, and that it is not possible to end the troubleshooting session early by accepting a penalty cost for the remaining faults. However, in [7] further variations of this heuristic are proposed that allow both imperfect repairs and the possibility to end troubleshooting sessions early.

### Upper Bound Heuristics

If it is possible to efficiently compute the expected cost of a known plan or policy that is guaranteed to solve the troubleshooting problem this can be used as an upper bound heuristic. Such a heuristic will be uniformly improvable (see Section 3.3.1). The heuristic $h_{heck}$ described in Section 2.2.1 is such a heuristic which can be used when there can be at most a single fault, action costs are constant, and it is possible to verify whether faults on the system are present or not (i.e. a system check) [62].

In [7] two novel upper bound heuristics are proposed. The first heuristic is for the case where multiple faults can be present and the action costs are state dependent (e.g. because they are macro actions as described in Section 3.3.2). The second heuristic is for the case when there can also be imperfect repairs (i.e. there is a probability that the fault remains after an attempted repair of it), it is possible to end the troubleshooting session early, and there is no system check available. Both heuristics are based on the idea of evaluating a plan consisting of a sequence of smaller partial plans each ensuring that a suspected fault is removed.

The first heuristic has two types of partial plans. The first one is for faults that can be inspected, i.e. there exists an action that can verify whether the fault is present or not, and the second one is for those faults that are not observable:

1. Inspect whether the fault is present or not. If it is not present, continue to the partial plan for the next fault. Otherwise, repair the fault and then perform a system check. If the system now is non-faulty, end the troubleshooting session, otherwise continue to the partial plan for the next fault.

2. Immediately repair the fault and then perform a system check. If the system now is non-faulty, end the troubleshooting session, otherwise continue to the partial plan for the next fault.

After each partial plan is executed, the troubleshooting session is either ended or it is continued with the next partial plan. In this way the full plan never branches out and therefore it is possible to compute its expected cost efficiently from any given belief state in time linear with the number of faults. The ordering of these partial plans is arbitrary for the sake of the heuristic being an upper bound, but the best performance has been experienced when they are ordered by their probability/inspection cost ratios as for the heuristic $h_{heck}$ described in Section 2.2.1.

When repairs are imperfect we cannot be sure that repairing a faulty component really removes the fault. Further if we cannot make a system check we cannot know if an attempted repaired has succeeded. However, if the troubleshooting session can be ended early, not all faults must be successfully repaired. The second heuristic uses five types of partial plans, each representing alternative methods of either verifying that a fault is not present or ignoring it until the troubleshooting session is ended. Each partial plan is evaluated for each fault and the partial plan with the lowest expected cost is selected. The last three partial plans can only be used for faults that can be inspected. The five partial plans are the following:

1. Immediately accept the risk that the fault is present (adding the expected penalty for this to the expected cost of this plan), then continue to the partial plan for the next fault.
2. Repair the fault, then continue with partial plan 1 for this fault.
3. Repair the fault and then inspect it. If the fault is still present, repeat these two actions until the fault is no longer observed to be present. Then continue to the partial plan for the next fault.
4. Inspect the fault. If it is not present continue to the partial plan for the next fault. Otherwise continue with partial plan 2.
5. Inspect the fault. If it is not present continue to the partial plan for the next fault. Otherwise continue with partial plan 3.

Like the previous one, this heuristic can also be evaluated in time linear in the number of faults. In experiments with a troubleshooting model of a hydraulic braking system, it is shown that this heuristic improves the quality of decisions computed using IBLAO* when planning time is limited [7].

### 3.3.5 Integrated Remote and Workshop Troubleshooting

In [11] a novel troubleshooting framework is presented for integrated remote and workshop troubleshooting. Then troubleshooting starts at the instant a fault is detected on the road and ends when the vehicle leaves the workshop. The types of actions that are available for planning are:

- *reservation actions* that make new reservations at specific workshops,
- *transport actions* that take the vehicle to the next planned workshop visit,
- *observing actions* (remote or in-workshop),

- *repair actions*, and
- the *end session action* that ends the troubleshooting session.

The planner needs to consider both the direct costs from making repairs and observations and the indirect costs from the vehicle downtime experienced by the vehicle owner. Making unnecessary visits to the workshop is costly for the vehicle owner. To reduce the overall expected cost, remote tests can be run on the vehicle so that diagnoses where the suspected faults are serious enough to require immediate attention are separated from those where the faults are such that it can be expected to be less costly to wait until the next already scheduled workshop visit than having to disrupt operations and schedule a new workshop visit.

Downtime costs are modeled into the cost functions of the reservation and transport actions. For a reservation action, the cost is a constant representing the additional costs the vehicle owner has because of disturbances in the work schedule when having to perform an unexpected workshop visit. The cost of transport actions depends on how the vehicle is transported to the next planned workshop visit. Three options are available. Either the vehicle is operated normally until the time of the next planned workshop visit, the vehicle is put on hold until the time of the next planned workshop visit and then it drives there, or the vehicle is towed to the workshop and then put on hold until the time of the visit. The cost of operating normally depends on the duration of the operation and the severity of present faults. The cost of being put on hold depends on the duration of the time being put on hold. The cost of towing depends on the distance to the workshop. To be able to correctly compute these costs, the state that previously only represented the diagnostic aspects of the vehicle is extended with variables that represent the time and location of the vehicle and the next planned workshop visit.

To be able to compute decisions with reasonable constraints on computation time, a new planning algorithm is needed. The new algorithm is a look-ahead search algorithm based on the algorithm of Langseth and Jensen [75] described in Section 2.2.1. It can compute suboptimal decisions in time linear in the number of actions and the time for computing and evaluating new states (which depends on the inference method used by the diagnoser and the heuristics used). If the vehicle is in the workshop the algorithm proceeds much like Langseth and Jensen's algorithm. However, when the vehicle is not in the workshop, it needs to reason about the best way of transporting the vehicle to the workshop and whether remote testing is needed. The cost model of transport actions is conceived so that the costs increase monotonically with the time and distance to the next workshop visit. Then it is only necessary to compare the cases where the vehicle is transported to the currently next planned workshop visit with those where the vehicle is transported to the nearest workshop at the earliest possible time.

The new algorithm can be used together with any of the previously described methods for implementing the diagnoser, abstracting away assembly and disassembly actions, and computing heuristics. For a clearer presentation, the diagnoser used in the paper is a conventional static Bayesian network.

## 3.4 Applications

During the development of the troubleshooting framework, three real subsystems of a heavy truck have been used as application examples: an auxiliary truck braking system called the *retarder*, a common-rail fuel injection system called the XPI system (eXtreme high Pressure fuel Injection system), and an engine temperature control and monitoring system. In this section, we will describe these systems and how they have been used in the different contributions.

### 3.4.1 The Retarder

The retarder is an auxiliary hydraulic braking system that allows braking of the truck without applying the conventional brakes. The system consists of a combination of mechanical, hydraulic, pneumatic, and electronic parts that are controlled by an electronic control unit (ECU). This is typical for systems on heavy vehicles. The most challenging faults to troubleshoot are internal oil leakages, air leakages, and certain mechanical faults that cannot be isolated by the on-board diagnostic system.

Figure 3.6 shows a schematic of the retarder. The central component of the retarder is the torus which consists of two parts, a rotor (1) and a stator (2). The rotor rotates with the engine and the stator is rigidly fixed with the retarder housing. When engaged, the volume in the torus is filled with oil causing friction between the rotor and stator which is converted to braking torque that is transferred via the retarder axle (3) to the propeller shaft (4) in the gear box. This friction heats up the oil which needs to be cooled off by circulating it through a cooler (8) using the pump (5). The amount of braking torque is proportional to the engine speed and the amount of oil in the system. At full effect and high engine speed, the retarder can generate braking torque of a magnitude that is comparable with the full engine torque. To engage the retarder, oil is taken from the oil sump and inserted into the system through the accumulator valve (12). Smaller adjustments of the amount of oil in the system are made using a control valve (6). To disengage the retarder, the safety valve (10) is opened and the torus is drained of oil. The valves are controlled by the ECU through a pneumatic valve block (7) using inputs from sensors that measure the coolant temperature (13), the oil temperature (14), and the oil pressure (15).

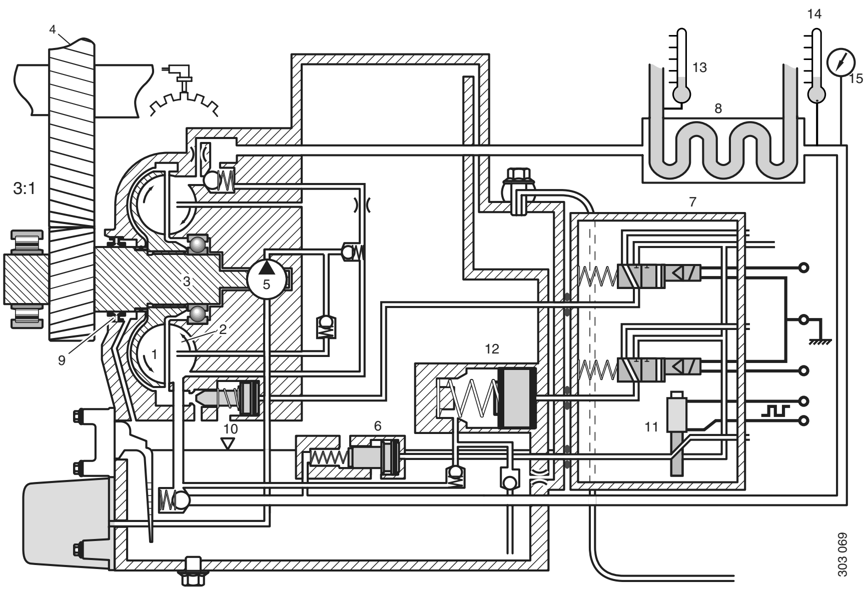

Figure 3.6: Schematic of the retarder

When modeling the retarder, the first step was to let experts identify all system components that were known to fail. Then the experts listed all observations that could occur because of a failure on a given component. For each dependency parameters were estimated by the experts. To reduce the number of parameters needing to be set, leaky noisy-OR distributions [63] were used to model CPT:s with many dependent variables. In total, the diagnostic model has 20 components which can be faulty and 25 observations. The planning model has 68 actions. Information about their costs, effects, and preconditions was obtained from the workshop manual [111] and a proprietary database containing standard times for all the actions. The assembly graph is modeled with 13 assembly variables using information from the workshop manual.

In [4, 5], a static Bayesian network model of the retarder is used and in [3, 7, 8] nsDBN models are used. The planning model is used in [4, 5, 7, 8]. A detailed presentation of the nsDBN model and its parameters can be found in [7].

## 3.4.2 The XPI System

The XPI system is responsible for dosing the correct amount of diesel fuel into the cylinders. It is a *common rail* fuel injection system, which means that highly pressurized fuel is stored in an accumulator called the common rail before being

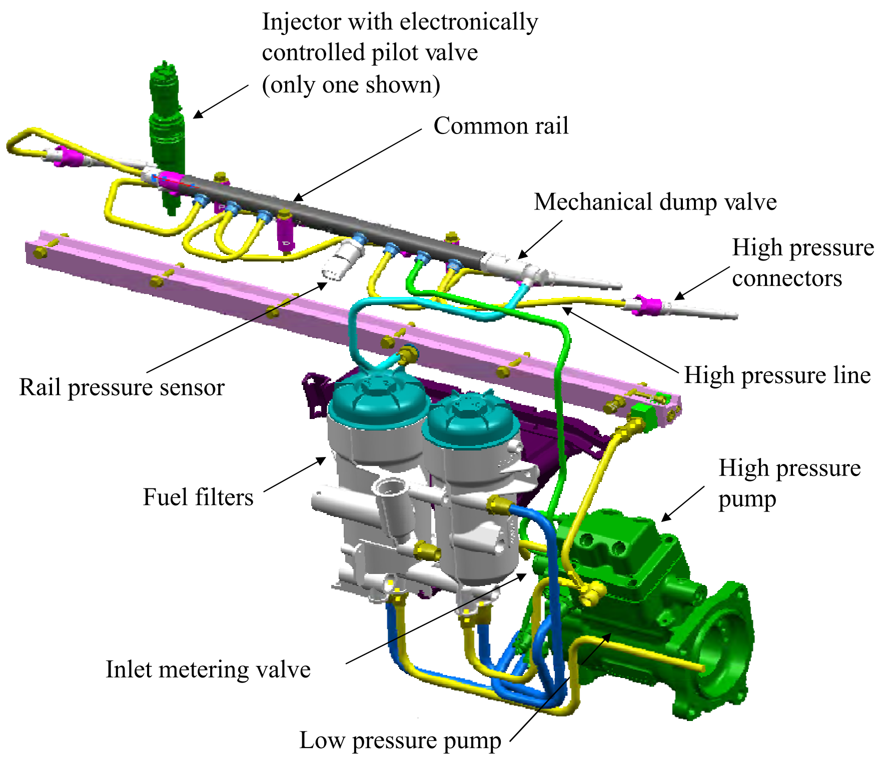

Figure 3.7: An overview of the XPI system and its main components.

injected into the cylinders. Using this high pressure, the fuel can be injected in multiple sprays with a high degree of control independently of the piston positions for maximized fuel efficiency and environmental performance.

Figure 3.7 shows an overview of the XPI system and its main components. The low pressure pump (LPP) circulates fuel from the fuel tank through the fuel filters. Then the inlet metering valve (IMV) directs some of this fuel to the high pressure pump (HPP) so that the fuel pressure in the common rail, as measured by the pressure sensor, is as commanded by the engine management system (EMS). From the common rail, fuel is directed to the injectors through a high pressure line and a connector. When commanded by the EMS, the injectors spray fuel into the cylinders through a small micrometer-sized valve. Excess fuel from the injectors is returned to the low pressure side through the return fuel rail. There is also a mechanical dump valve that protects the common rail from excessive pressures by releasing fuel into the return fuel rail.

The on-board diagnosis system on the vehicle is very capable of detecting and isolating *electrical* faults on the injectors, the IMV, and the rail pressure sensor. However *mechanical* faults and leakage problems are typically only detectable by observing that it is not possible to gain the commanded pressure in the common rail. If this happens a diagnostic trouble code (DTC) is generated that indicates that the rail pressure is below the commanded pressure. Some of the faults that are indistinguishable by the on-board diagnostic system can be distinguished by additional testing either remotely or at the workshop. For example, a mechanic can measure the return fuel pressure by mounting a pressure meter on the return fuel line or measure the output effect of the cylinders by running a test on the EMS. Other faults can only be distinguished by trial and error. However, the mechanic must take care when troubleshooting the XPI system because many components on the high pressure side must be completely replaced with new components if they are removed because of strict cleanliness requirements. Because of this, the consequences of performing unnecessary actions on the high pressure side can be costly.

A diagnostic model of the XPI system is used in the diagnostic framework for integrated remote and workshop troubleshooting described in Section 3.2.2. This model has been developed together with domain experts and it has 19 components, 47 faults, 86 symptoms, and 89 observations. Also, 527 symptom cause variables and 15 auxiliary variables were needed to model all dependencies between faults, symptoms, and observations.

Of the observations, 36 variables correspond to DTC:s that can be generated by the on-board diagnostic system. The behavior of the DTC:s is well documented with regard to which faults they are sensitive for. 9 observations are for symptoms that a driver can experience. Some of these are vague and can be caused by many faults, e.g. "high fuel consumption" and "loss of function", which makes it difficult for the expert to correctly assess the parameters. However, these symptoms are often reported in the warranty claims and can be learned from that source instead. The remaining 44 observations are for tests that can be run in the workshop. The parameters for these cannot be learned from the warranty data. However the workshop tests are developed by in-house engineers which makes it easier to set parameter values with great confidence.

This model was validated by simulating the instructions from a troubleshooting tree in the workshop manual used by the mechanics and comparing whether the output of the diagnoser agrees with the information found in the manual. Along all paths in the troubleshooting tree, the diagnoser assigns the highest probabilities to the components listed as possible explanations in the manual.

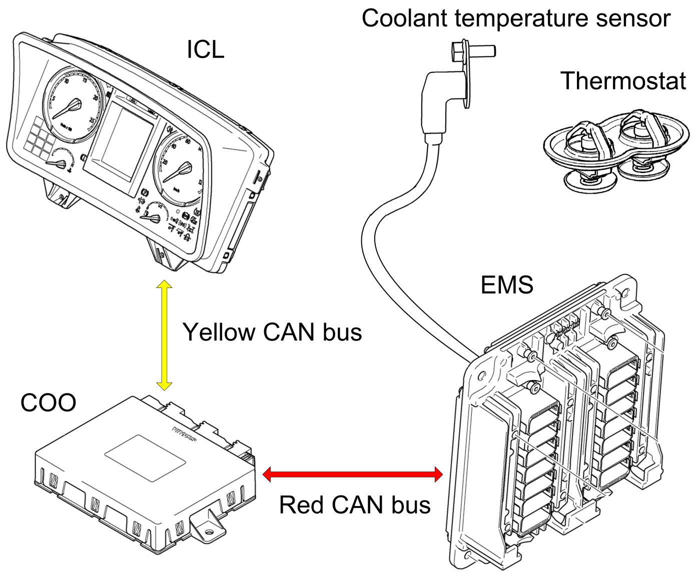

Figure 3.8: An overview of the engine temperature control and monitoring application example [11].

Another model of the XPI system is also used in [1, 2]. This is a different model than the above, where instead the diagnostic model is a static Bayesian network. This model was developed by students in a series of master-theses [79, 85, 133, 144].

### 3.4.3 Engine Temperature Control and Monitoring System

A small application example of the engine temperature control and monitoring system is used in [11]. The purpose of this application example is to demonstrate the concept of integrated remote and workshop troubleshooting on a real truck in a setting where a useful remote diagnostic test could be implemented on a vehicle that is unprepared for remote diagnosis without making invasive modifications. In the scenario used in the paper, the driver has called the help desk after observing that the engine temperature appears low despite that the engine has been running for some time. Nothing else that could indicate problems has been observed, and a decision needs to be taken whether it is safe to continue driving until the next scheduled maintenance occasion or if it could be a serious fault requiring immediate attention.

An overview of the system is shown in Figure 3.8. The components that can cause this problem are the coolant temperature sensor, the thermostat, the engine control unit (EMS), the coordinator control unit (COO), the instrument cluster

(ICL), and two data communication buses, the red CAN bus (used by the EMS), and the yellow CAN bus (used by the ICL). The thermostat uses a bimetal to regulate the engine temperature by opening a valve when the engine reaches operating temperature. If it is stuck open, the engine cannot attain operating temperature. The engine temperature as measured by the temperature sensor is relayed via the EMS, the red CAN bus, the COO, and the yellow CAN bus to the ICL where it is presented to the driver on the temperature gauge. A fault at either place in this chain can cause the temperature to appear low. A fault on the ICL that prevents it from displaying the correct engine temperature is considered to be a minor fault that does not require immediate attention. A remote test that sweeps the gauges in the ICL can discriminate between such a fault and other more serious faults on the engine.

The resulting diagnostic model is a static Bayesian network with 57 variables of which 14 represent different faults that can cause the observed problem, and 22 represent observations that can be seen together with any of these faults. There are 27 actions of which there are 7 repair actions, 8 observing actions, 3 reservation actions, and 9 transport actions. Figure 1.4 shows the optimal troubleshooting plan for the scenario described above when the next scheduled workshop visit is in 11 days.

# Chapter 4

# Conclusions

The purpose of this thesis has been to study and develop methods for computer-assisted troubleshooting of heavy vehicles such as trucks and buses. In this type of troubleshooting, the person that is troubleshooting a vehicle problem is assisted by a computer that is capable of listing possible faults that can explain the problem and gives recommendations of which actions to take in order to solve the problem so that the expected cost of restoring the vehicle is low. To achieve this, such a system must be capable of solving two tasks: the diagnosis problem of finding which the possible faults are and the planning problem of deciding which action should be taken.

The diagnosis problem has been approached using Bayesian network models. Diagnostic frameworks using event-driven non-stationary dynamic Bayesian networks have been developed for diagnosing the vehicle during troubleshooting. The frameworks can be applied in settings where either the vehicle is in the workshop only, or the troubleshooting needs to start as soon as a problem is detected on the road in which case the vehicle is monitored during longer periods of time. The models can be created using expert knowledge and the model parameters can be tuned using statistical data. It is shown how necessary operations such as computing fault and outcome probabilities can be done efficiently. The models are modular in the same way as the vehicles are, and therefore models of complete vehicles can be made specific for each vehicle while each individual submodel can be created and trained independently.

For the planning problem, planners have been created that select actions so that the expected cost of repairing the vehicle is minimized. For this new methods have been developed for increasing the efficiency of existing planning algorithms. By separating diagnostic actions such as tests and repairs from other preparatory actions, the number of steps required to reach a goal state in the planners' search graph can become smaller. This has been shown specifically for the troubleshooting problem and more generally for a larger class of problems. New search heuristics provide better lower and upper bound estimates of the minimal expected cost allowing the planners to converge faster. Also completely new planning algorithms have been developed. The new algorithm Iterative Bounding LAO* can compute action recommendations with provable bounds on the optimal expected cost when it is formulated as a stochastic shortest path problem. It uses a smaller search graph when compared with other comparable existing algorithms. For the case when remote is integrated with the workshop diagnosis, the planner also needs to consider the problem of deciding when and where to visit a workshop. A novel algorithm has been developed that in linear time can compute recommendations of actions in this setting.

The theory developed for this thesis has been evaluated on models of subsystems that are characteristic for troubleshooting heavy vehicles: an auxiliary braking system, a fuel injection system, and an engine temperature control and monitoring system.

The theoretical methods presented in this thesis are believed to be mature enough to be evaluated in a larger scale on real vehicles in real workshops. Fully functioning, such a system for computer-assisted troubleshooting can generate large cost savings for both workshops and vehicle owners. This is also important for the vehicle manufacturers in order to remain competitive in the future.

# References

[1] Håkan Warnquist, Mattias Nyberg, and Petter Säby. Troubleshooting when Action Costs are Dependent with Application to a Truck Engine. In *Proceedings of the 10th Scandinavian Conference on Artificial Intelligence (SCAI'08)*, 2008.

[2] Håkan Warnquist and Mattias Nyberg. A Heuristic for Near-Optimal Troubleshooting Using AO*. In *Proceedings of the 19th International Workshop on the Principles of Diagnosis (DX'08)*, 2008.

[3] Anna Pernestål, Håkan Warnquist, and Mattias Nyberg. Modeling and Troubleshooting with Interventions Applied to an Auxiliary Truck Braking System. In *Proceedings of 2nd IFAC workshop on Dependable Control of Discrete Systems (DCDS'09)*, 2009.

[4] Håkan Warnquist, Anna Pernestål, and Mattias Nyberg. Anytime Near-Optimal Troubleshooting Applied to a Auxiliary Truck Braking System. In *Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'09)*, 2009.

[5] Håkan Warnquist, Jonas Kvarnström, and Patrick Doherty. Planning as Heuristic Search for Incremental Fault Diagnosis and Repair. In *Proceedings of the 2nd Scheduling and Planning Applications woRKshop (SPARK'09)*, 2009.

[6] Håkan Warnquist, Jonas Kvarnström, and Patrick Doherty. Iterative Bounding LAO*. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, 2010.

[7] Håkan Warnquist. Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis. Licentiate Thesis, Linköping University, 2011.

[8] Anna Pernestål, Håkan Warnquist, and Mattias Nyberg. Modeling and inference for troubleshooting with interventions applied to a heavy truck auxiliary braking system. *Engineering Applications of Artificial Intelligence*, 25(4):705 – 719, 2012.

[9] Håkan Warnquist, Jonas Kvarnström, and Patrick Doherty. Exploiting Fully Observable and Deterministic Structures in Goal POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 13)*, 2013.

[10] Tony Lindgren, Håkan Warnquist, and Martin Eineborg. Improving the maintenance planning of heavy trucks using constraint programming. In *Proceedings of the Twelfth International Workshop on Constraint Modelling and Reformulation (ModRef 2013)*, pages 74–90. Université Laval, 2013.

[11] Håkan Warnquist, Mattias Nyberg, and Jonas Biteus. Guided Integrated Remote and Workshop Troubleshooting of Heavy Trucks. *SAE International Journal of Commercial Vehicles*, 7(1):25–36, 04 2014.

[12] Håkan Warnquist, Jonas Kvarnström, and Patrick Doherty. A Modeling Framework for Troubleshooting Automotive Systems, 2015. Manuscript submitted to Applied Artificial Intelligence.

[13] Mauricio Araya-Lòpez, Vincent Thomas, Olivier Buffet, and François Charpillet. A Closer Look at MOMDPs. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence*, 2010.

[14] Joaquim Armengol, Anibal Bregon, Teresa Escobet, Esteban R. Gelso, Mattias Krysander, Mattias Nyberg, Xavier Olive, B. Pulido, and Lousie Trave-Massuyes. Minimal Structurally Overdetermined Sets for Residual Generation: A Comparison of Alternative Approaches. In *Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'09)*, Barcelona, Spain, 2009.

[15] K. J. Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1): 174 – 205, 1965.

[16] Stefania Bandini, Ettore Colombo, Giuseppe Frisoni, Fabio Sartori, and Joakim Svensson. Case-Based Troubleshooting in the Automotive Context: The SMMART Project. In *Proceedings of the 9th European conference on Advances in Case-Based Reasoning (ECCBR'08)*, 2008.

[17] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2): 81–138, 1995.

[18] Michèle Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[19] Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[20] E. Benazera and E. Chanthery. The Challenge of Solving POMDPs for Control, Monitoring and Repair of Complex Systems. In *Proceedings of the 19th International Workshop on Principles of Diagnosis (DX'08)*, 2008.

[21] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics in Operation Research*, 16:580–595, 1991.

[22] Mogens Blanke, Michel Kinnaert, Jan Lunze, Marcel Staroswiecki, and J. Schröder. *Diagnosis and Fault-Tolerant Control.* Springer Verlag, New York, 2006.

[23] Andrea Bobbio, Luigi Portinale, Michele Minichino, and Ester Ciancamerla. Improving the analysis of dependable systems by mapping fault trees into bayesian networks. *Reliability Engineering & System Safety*, 71 (3):249–260, 2001.

[24] Blai Bonet and Hector Geffner. Solving POMDPs: RTDP-Bel vs. Point-based Algorithms. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, 2009.

[25] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the 13th National Conference on Artificial Intelligence*, 1996.

[26] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 33–42. Morgan Kaufmann Publishers Inc., 1998.

[27] Johnn S Breese, Eric J Horvitz, Mark A Peot, Rodney Gay, and George H Quentin. Automated decision-analytic diagnosis of thermal performance in gas turbines. In *Proceedings of the International Gas Turbine and Aero-engine Congress and Exposition, Cologne, Germany, American Society of Mechanical Engineers*, 1992.

[28] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1-2): 99–134, 1998.

[29] E Chanthery, Y Pencole, and N Bussac. An AO*-like algorithm implementation for active diagnosis. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, pages 378–385, 2010.

[30] Rahul Chougule, Dnyanesh Rajpathak, and Pulak Bandyopadhyay. An integrated framework for effective service and repair in the automotive domain: An application of association mining and case-based-reasoning. *Computers in Industry*, 62(7):742 – 754, 2011.

[31] E. Chow and AS. Willsky. Analytical redundancy and the design of robust failure detection systems. *IEEE Transactions on Automatic Control*, 29(7): 603–614, Jul 1984.

[32] Gregory F Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2):393–405, 1990.

[33] Marie-Odile Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(5):2163–2177, 2004.

[34] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[35] Daimler AG. From hard haul to high-tech: 50 years of truck development for the sake of the environment, safety, comfort and economy. Press release, http://www.media.daimler.com, April 2010.

[36] Nando De Freitas, Richard Dearden, Frank Hutter, Ruben Morales-Menendez, Jim Mutch, and David Poole. Diagnosis by a waiter and a mars explorer. *Proceedings of the IEEE*, 92(3):455–468, 2004.

[37] Johan de Kleer and Brian C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[38] Johan de Kleer and Brian C. Williams. Diagnosis with Behavioral Modes. In *Readings in Model-based Diagnosis*, pages 124–130, San Francisco, 1992. Morgan Kaufmann.

[39] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing Diagnoses and Systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[40] Johan de Kleer, Olivier Raiman, and Mark Shirley. One step lookahead is pretty good. In *Readings in Model-based Diagnosis*, pages 138–142. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[41] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1990.

[42] Decision Systems Laboratory of University of Pittsburg. SMILE reasoning engine and GeNie modeling environment. Retrieved from http://genie.sis.pitt.edu, May 2014.

[43] Mark Devaney and William Cheetham. Case-Based Reasoning for Gas Turbine Diagnostics. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS'05)*, 2005.

[44] DFF International. RHA Cost Tables 2014. Retrieved from, http://dffintl.co.uk/Cost_Tables_2014.pdf, May 2014.

[45] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification*. Wiley, second edition, 2001.

[46] Alexander Feldman, Gregory Provan, and Arjan Van Gemund. FRACTAL: Efficient fault isolation using active testing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 778–784, 2009.

[47] Gerhard Friedrich and Wolfgang Nejdl. Choosing Observations and Actions in Model-Based Diagnosis/Repair Systems. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning (KR'92)*, 1992.

[48] Alvaro García and Eduardo Gilabert. Mapping FMEA into Bayesian networks. *International Journal of Performability Engineering*, 7(6):525–537, 2011.

[49] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.

[50] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[51] Sahika Genc and Stéphane Lafortune. Distributed diagnosis of discrete-event systems using Petri nets. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN'03)*, 2003.

[52] Eric Georgin, Frederic Bordin, S. Loesel, and Jim R. McDonald. CBR Applied to Fault Diagnosis on Steam Turbines. In *Proceedings of the First United Kingdom Workshop on Progress in Case-Based Reasoning*, 1995.

[53] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140(2):107–113, 1993.

[54] Alban Grastien, Anbulagan, Jussi Rintanen, and Elena Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1, pages 305–310, 2007.

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

[56] Anton Gustafsson and Sebastian Wassberg. Ekonomiska konsekvenser till följd av oplanerade stillestånd – En multipel fallstudie av företag i den svenska åkeribranschen, 2013.

[57] Fredrik Gustafsson. *Adaptive filtering and change detection*, volume 1. Wiley New York, 2000.

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

[59] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[60] W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[61] David Heckerman. A Tractable Inference Algorithm for Diagnosing Multiple Diseases. In *Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence (UAI'90)*, 1990.

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

[63] Max Henrion. Some Practical Issues in Constructing Belief Networks. In *Proceedings of the 3rd Conference on Uncertainty in Artificial Intelligence (UAI'87)*, 1987.

[64] Rolf Isermann. Model-based fault detection and diagnosis: status and applications. In *Proceedings of the 16th IFAC Symposium on Automatic Control in Aerospace (ACA'04)*, 2004.

[65] Tsukasa Ishigaki, Tomoyuki Higuchi, and Kajiro Watanabe. Spectrum classification for early fault diagnosis of the LP gas pressure regulator based on the Kullback-Leibler kernel. In *Proceedings of the 16th IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing*, pages 453–458. IEEE, 2006.

[66] Lindsay B. Jack and Asoke K. Nandi. Fault detection using support vector machines and artificial neural networks augmented by genetic algorithms. *Mechanical Systems and Signal Processing*, 16(2-3):373–390, 2002.

[67] Finn V Jensen, Uffe Kjærulff, Brian Kristiansen, Helge Langseth, Claus Skaanning, Jirí Vomlel, and Marta Vomlelová. The SACSO methodology for troubleshooting complex systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 15(04):321–333, 2001.

[68] Carl M Kadie, David Hovel, and Eric Horvitz. MSBNx: A component-centric toolkit for modeling and inference with Bayesian networks. *Microsoft Research, Richmond, WA, Technical Report MSR-TR-2001-67*, 28, 2001.

[69] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82 (Series D):35–45, 1960.

[70] Daphne Koller and Avi Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI'97)*, pages 302–313. Morgan Kaufmann Publishers Inc., 1997.

[71] B. Koppen-Seliger and P. M. Frank. Fault detection and isolation in technical processes with neural networks. In *Proceedings of the 34th IEEE Conference on Decision and Control (CDC'95)*, pages 2414–2419 vol.3, 1995.

[72] Olaf Krieger. *Wahrscheinlichkeitsbasierte Fahrzeugdiagnose mit individueller Prüfstrategie*. PhD thesis, Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2011.

[73] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Proceedings of the 2008 Robotics: Science and Systems Conference*, 2008.

[74] Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund, and Alexander Feldman. First International Diagnosis Competition - DXC'09. In *Proceedings of the 20th International Workshop on Principles of Diagnosis (DX'09)*, 2009.

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

[76] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[77] Mario Lenz, Hans-Dieter Burkhard, Petra Pirk, Eric Auriol, and Michel Manago. CBR for diagnosis and decision support. *AI Commun.*, 9(3): 138–146, 1996.

[78] Juan Liu, Lukas Kuhn, and Johan De Kleer. Computationally efficient tiered inference for multiple fault diagnosis. *Proceedings of Prognostics and Health Management (PHM2009)*, 2009.

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

[80] Anders L Madsen, Frank Jensen, Uffe B Kjaerulff, and Michael Lang. The hugin tool for probabilistic graphical models. *International Journal on Artificial Intelligence Tools*, 14(03):507–543, 2005.

[81] MAN Truck International. MAN service agreements. Retrieved from http://www.truck.man.eu/global/en/services-and-parts/maintenance-and-parts/service-contracts/Service-contracts.html, May 2014.

[82] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5 (4):115–133, 1943.

[83] H. Brendan Mcmahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, 2005.

[84] Merriam-Webster. Diagnosis. Retrieved from http://www.merriam-webster.com/dictionary/diagnosis, July 2014.

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

[86] Sriram Narasimhan, Richard Dearden, and Emmanuel Benazera. Combining particle filters and consistency-based approaches for monitoring and diagnosis of stochastic hybrid systems. In *Proceedings of the 15th International Workshop on Principles of Diagnosis (DX'04)*, 2004.

[87] Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, San Francisco, 1980.

[88] Mattias Nyberg. A Generalized Minimal Hitting-Set Algorithm to Handle Diagnosis With Behavioral Modes. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 41(1):137–148, 2011.

[89] Thorsten J. Ottosen and Finn V. Jensen. When to test? Troubleshooting with postponed system test. *Expert Systems with Applications*, 38(10): 12142 – 12150, 2011.

[90] Thorsten Jørgen Ottosen and Finn V Jensen. The Cost of Troubleshooting Cost Clusters with Inside Information. In *26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, 2010.

[91] John Pandremenos, John Paralikas, Konstantinos Salonitis, and George Chryssolouris. Modularity concepts for the automotive industry: A critical review . *CIRP Journal of Manufacturing Science and Technology*, 1(3):148 – 152, 2009.

[92] Judea Pearl. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In *Proceedings of The 2nd National Conference on Artificial Intelligence (AAAI'82)*, 1982.

[93] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society*, 1985.

[94] Judea Pearl. *Causality*. Cambridge University Press, 2000.

[95] Yannick Pencolé and Marie-Odile Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164(1-2): 121–170, 2005.

[96] Anna Pernestål. *Probabilistic Fault Diagnosis with Automotive Applications*. PhD thesis, Linköping University, Vehicular Systems, The Institute of Technology, 2009.

[97] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 2003.

[98] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[99] B. W. Ricks and O. J. Mengshoel. The Diagnostic Challenge Competition: Probabilistic Techniques for Fault Diagnosis in Electrical Power Systems. In *Proceedings of the 20th International Workshop on Principles of Diagnosis (DX'09)*, Stockholm, Sweden, 2009.

[100] Jussi Rintanen. Complexity of Planning with Partial Observability. In *Proceedings of the 14th International Conference on Automated Planning (ICAPS'04)*, 2004.

[101] Irina Rish, Mark Brodie, Sheng Ma, Natalia Odintsova, Alina Beygelzimer, Genady Grabarnik, and Karina Hernandez. Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks*, 16(5):1088–1109, 2005.

[102] Joshua W. Robinson and Alexander J. Hartemink. Non-stationary dynamic Bayesian networks. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS'08)*, pages 1369–1376, 2008.

[103] Indranil Roychoudhury, Gautam Biswas, and Xenofon Koutsoukos. A Bayesian approach to efficient diagnosis of incipient faults. In *Proceedings 17th International Workshop on the Principles of Diagnosis (DX'06)*, 2006.

[104] Indranil Roychoudhury, Gautam Biswas, and Xenofon Koutsoukos. Distributed Diagnosis of Dynamic Systems Using Dynamic Bayesian Networks. In *Proceedings 20th International Workshop on the Principles of Diagnosis (DX'09)*, 2009.

[105] Indranil Roychoudhury, Gautam Biswas, and Xenofon Koutsoukos. Designing distributed diagnosers for complex continuous systems. *IEEE Transactions on Automation Science and Engineering*, 6(2):277–290, 2009.

[106] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[107] M Saadawia and Dirk Söffker. Svm-based fault diagnosis system for materials change detection. In *Proceedings of the 7th Workshop on Advanced Control and Diagnosis (ACD'09)*, pages 19–20, 2009.

[108] Scott Sanner, Robby Goetschalckx, Kurt Driessens, and Guy Shani. Bayesian Real-Time Dynamic Programming. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1784–1789, 2009.

[109] U. K. Sarkar, P. P. Chakrabarti, S. Ghose, and S. C. Desarkar. Improving Greedy Algorithms by Lookahead-Search. *Journal of Algorithms*, 16(1): 1–23, 1994.

[110] C. Saunders, A. Gammerman, H. Brown, and G. Donald. Application of Support Vector Machines to Fault Diagnosis and Automated Repair. In *Proceedings of the 11th International Workshop on the Principles of Diagnosis (DX'00)*, 2000.

[111] Scania CV. Scania Multi Service. Retrieved from https://mppv.scania.com/Site/, November 2010.

[112] Scania CV. Taking a look back at the development of Scania Retarder. http://newsroom.scania.com/en-group/2013/02/22/a-brake-with-history/, February 2013.

[113] Scania CV. Scania Adaptive Cruise Control (ACC). Retrieved from http://www.scania.com/products-services/trucks/safety-driver-support/driver-support-systems/acc/, April 2014.

[114] Scania CV. Scania EGR. Retrieved from http://www.scania.com/products-services/trucks/main-components/engines/engine-technology/egr/, June 2014.

[115] Scania CV. Scania Repair & Maintenance contract. Retrieved from http://www.scania.com/products-services/services/workshop-services/, May 2014.

[116] Scania CV. Scania Repair & Maintenance contract. Retrieved from http://www.scania.co.uk/services/support-programmes/repair-maintenance/, May 2014.

[117] Scania CV. Scania SCR. Retrieved from http://www.scania.com/products-services/trucks/main-components/engines/engine-technology/scr/, June 2014.

[118] Scania CV. Scania SDP3. Retrieved from http://tis.scania.com/site/Products.aspx?Category=SDP#, June 2014.

[119] Matthew L Schwall and J Christian Gerdes. A probabilistic approach to residual processing for vehicle fault detection. In *Proceedings of the American Control Conference*, volume 3, pages 2552–2557. IEEE, 2002.

[120] Peter Söderholm. A system view of the No Fault Found (NFF) phenomenon. *Reliability Engineering & System Safety*, 92(1):1 – 14, 2007. ISSN 0951-8320.

[121] Guy Shani, Ronen I. Brafman, and Solomon E. Shimony. Forward search value iteration for POMDPs. In *In Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

[122] Tomi Silander and Petri Myllymäki. A Simple Approach for Finding the Globally Optimal Bayesian Network Structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI'96)*, 2006.

[123] Satnam Singh, Halasya Siva Subramania, Steven W Holland, and Jason T Davis. Decision Forest for Root Cause Analysis of Intermittent Faults. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(6):1818–1827, 2012.

[124] Claus Skaaning. First commercial bayesian software for intelligent troubleshooting and diagnostics. *ERCIM NEWS*, 56:20–21, 2004.

[125] Trey Smith and Reid G. Simmons. Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004.

[126] Trey Smith and Reid G. Simmons. Point-Based POMDP Algorithms: Improved Analysis and Implementation. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, 2005.

[127] Trey Smith and Reid G. Simmons. Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, 2006.

[128] Edward Sondik. *The optimal control of partially observable Markov processes.* PhD thesis, Stanford, 1971.

[129] Matthijs T. J. Spaan and Nikos A. Vlassis. Perseus: Randomized Point-based Value Iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[130] SAE Standards. J2012: Diagnostic Trouble Code definitions, 2013.

[131] Angus Stevenson, editor. *Oxford Dictionary of English.* Oxford University Press, third edition, 2010.

[132] Ying Sun and Daniel S. Weld. A framework for model-based repair. In *Proceedings of 11th National Conference on Artificial Intelligence (AAAI'93)*, 1993.

[133] Helena Sundberg. Decision-theoretic troubleshooting using bayesian networks - guided troubleshooting of a diesel injection system. Master's thesis, School of Computer Science and Communication, Royal Institute of Technology, 2008.

[134] M. Svensson, S. Byttner, and T. Rognvaldsson. Self-organizing maps for automatic fault detection in a vehicle cooling system. In *In Proceedings of the 4th International IEEE Conference on Intelligent Systems (IS'08)*, volume 3, pages 8–12, 2008.

[135] Carl Svärd, Mattias Nyberg, Erik Frisk, and Mattias Krysander. Automotive engine {FDI} by application of an automated model-based and data-driven design methodology. *Control Engineering Practice*, 21(4):455 – 472, 2013.

[136] Louise Travé-Massuyès. Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, 27(0):1 – 16, 2014.

[137] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293 – 311, 2003.

[138] Vandi Verma, Geoff Gordon, Reid Simmons, and Sebastian Thrun. Particle Filters for Rover Fault Diagnosis. *IEEE Robotics & Automation Magazine Special Issue on Human Centered Robotics and Dependability*, 2004.

[139] Sylvain Verron, Philippe Weber, Didier Theilliol, Teodor Tiplica, Abdessamad Kobi, Christophe Aubrun, et al. Decision with Bayesian network in the concurrent faults event. In *Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'09)*, 2009.

[140] Volvo Trucks. Volvo adds EGR exhaust gas recirculation to its successful 13-litre engine series . Press release, http://www.volvotrucks.com, April 2007.

[141] Volvo Trucks. Diagnostics. Retrieved from http://www.volvotrucks.com/trucks/na/en-us/parts_service/diagnostics/Pages/diagnostics.aspx, June 2014.

[142] Volvo Trucks. Service and maintenance agreements. Retrieved from http://www.volvotrucks.com/trucks/global/en-gb/trucks/services, May 2014.

[143] Volvo Trucks. Volvo Uptime Assurance. Retrieved from http://www.volvotrucks.com/trucks/global/en-gb/trucks/services/VAS/Pages/Volvo_uptime_assurance.aspx, May 2014.

[144] Håkan Warnquist and Petter Säby. Conditional planning for troubleshooting and repair in a partially observable environment. Master's thesis, Department of Computer and Information Science, Linköping University, 2008.

[145] Philippe Weber, Didier Theilliol, Christophe Aubrun, and Alexandre Evsukoff. Increasing effectiveness of model-based fault diagnosis: A dynamic Bayesian network design for decision making. In *Proceedings of 6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'07)*, pages 90–95, 2007.

[146] Jason D. Williams. Applying POMDPs to dialog systems in the troubleshooting domain. In *Proceedings of the Workshop on Bridging the Gap (NAACL-HLT '07)*, 2007.

[147] Bo-Suk Yang, Tian Han, and Yong-Su Kim. Integration of ART-Kohonen neural network and case-based reasoning for intelligent fault diagnosis. *Expert Systems with Applications*, 26(3):387–395, 2004.

[148] Changhe Yuan and Marek J Druzdzel. An importance sampling algorithm based on evidence pre-propagation. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI'02)*, pages 624–631, 2002.

[149] Wonham W. M. Zad S. H., Kwong R. H. Fault Diagnosis in Discrete Event Systems: Framework and model reduction. *IEEE Transactions on Automatic Control*, 48(7):1199–1212, 2003.

[150] Alice X. Zheng, Irina Rish, and Alina Beygelzimer. Efficient test selection in active diagnosis via entropy approximation. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 675–682, 2005.

# Publications

# Publications

Department of Computer and Information Science
Linköpings universitet

**Dissertations**

**Linköping Studies in Science and Technology**
**Linköping Studies in Arts and Science**
*Linköping Studies in Statistics*
*Linköpings Studies in Information Science*

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing – An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.