A Heuristic for Near-Optimal Troubleshooting Using AO*

Håkan Warnquist[*]
Dept. of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden
g-hakwa@ida.liu.se

Mattias Nyberg[*]
Dept. of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden
mattias.nyberg@scania.com

Abstract

When troubleshooting malfunctioning technical equipment, the task is to locate faults and make repairs until the equipment functions properly again. The AO* algorithm can be used to find troubleshooting strategies that are optimal in the sense that the expected cost of repair is minimal. We have adapted the AO* algorithm for troubleshooting in the automotive domain with limited time. We propose a new heuristic based on entropy. By using this heuristic, near-optimal strategies can be found within a fixed time limit. This is shown in empirical studies on a fuel injection system of a truck. In these results, the AO* algorithm using the new heuristic, performs better than other troubleshooting algorithms.

1 Introduction

The task of a troubleshooter is to locate faults and make repairs in machinery and technical equipment. For example, in the automotive domain, the troubleshooter is a mechanic that resolves a problem on a vehicle by repairing components. To know which components to repair, the mechanic can make observations to locate the cause of problem. As vehicles get more complex, the troubleshooting task becomes more difficult. An automated troubleshooter infers probabilities of component faults given observations and aids the mechanic by recommending appropriate actions to perform.

If we also want to minimize the costs, the troubleshooting task becomes an optimization problem that can be solved as a planning problem. This problem is known to be NP-hard so if we want the computations to be made while the mechanic is waiting, approximate methods are needed [Vomlelová and Vomlel, 2000].

AO* is a well known heuristic search algorithm that can solve planning problems involving uncertainty and feedback. It uses a heuristic function to focus its search. Provided that the heuristic is admissible the solution found is optimal [Nilsson, 1980]. There are heuristics that have been used with AO* to solve the troubleshooting problem in the literature [Faure, 2001; Vomlelová and Vomlel, 2000; Raghavan *et al.*, 1999], but we need stronger heuristics to focus the search more to find solutions of larger problems.

[*]Both authors are affiliated with Scania CV AB.

In this paper we propose a way to use AO* to find near-optimal solutions in a limited time. We use a new heuristic that, in empirical experiments on a case study, is shown to find solutions with lower costs compared to heuristics used in [Raghavan *et al.*, 1999; Vomlelová and Vomlel, 2000]. The case study is from the automotive domain and it is the fuel injection system of a truck.

In Section 2 we give the problem formulation and in Section 3 we present the case study. In Section 4 we describe some solution methods from the literature to the troubleshooting problem and other related problems. In Section 5 we show how AO* is used to solve the troubleshooting problem. In Section 6 we present the new heuristic. In Section 7 we do an empirical evaluation of the implementation and the new heuristic and finally we conclude in Section 8.

2 Problem Formulation

We will use the problem formulations of the troubleshooting frameworks presented in [Heckerman *et al.*, 1995], [Vomlelová and Vomlel, 2000], and [Langseth and Jensen, 2002]. In the troubleshooting problem the probabilistic relationships between component faults and observations is represented as a Bayesian network [Jensen, 1996]. With this probabilistic model it is possible to infer probability distributions of component faults conditioned on the information gained from previously performed actions.

Actions that we can perform on the system are either *repair actions* that successfully repair a single component or *observing actions* that observe the value of a node in the Bayesian network. Each action a is associated with a cost c_a. This cost is independent of any previously performed actions. In the Bayesian network there exists a single *problem-defining node* that indicates if *any* component is non-functioning. The observing action that observes this node is called the *function control*. The troubleshooting is said to be *successful* when a successful function control is made and the problem-defining node is observed to be non-indicating.

A *troubleshooting strategy* is defined as a labeled directed tree that describes the process of performing actions until the process terminates. The edges of the directed tree are labeled with actions. Edges labeled with observing actions are associated with a specific outcome of the action. Branching occurs only from nodes where each outgoing edge is labeled with an observing action. An example of a troubleshooting strategy is shown in Figure 1.

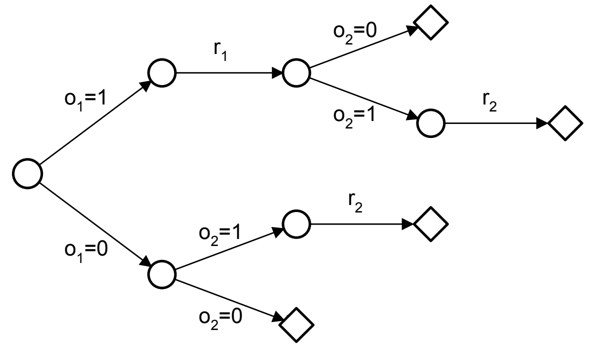

Figure 1: Example of a troubleshooting strategy with repair actions r_1, r_2 and observing actions o_1, o_2 having binary outcomes 0 or 1.

When a troubleshooting strategy is executed, we perform actions on the path from the root to a terminal node resulting from the responses of the observing actions. A terminal node is said to be *successful* if the action on the last edge is a successful function control and a *successful troubleshooting strategy* is a troubleshooting strategy where all terminal nodes are successful. This is done to validate that the problem is resolved.

The cost of reaching a terminal node t from the root node of a troubleshooting strategy s, $CR(s,t)$, is

$$CR(s,t) = \sum_{a \in \mathcal{P}(s,t)} c_a \qquad (1)$$

where $\mathcal{P}(s,t)$ is the set of all actions on the path from the root of the strategy s to the node t and c_a is the cost of the action a.

Let $\mathcal{T}(s)$ be the set of all terminal nodes of the troubleshooting strategy s and let ϵ be the current evidence representing our accumulated knowledge of the system, i.e. the results of all previously performed actions. When s is executed a terminal node $t \in \mathcal{T}(s)$ is reached with a certain probability, $P(T = t|s,\epsilon)$, where T is a stochastic variable with the outcome space $\mathcal{T}(s)$. These probabilities can be obtained from the probabilistic model. The *expected cost of repair* of s given ϵ, $ECR(s,\epsilon)$, is the expectation of the cost of reaching any node in $\mathcal{T}(s)$:

$$\begin{aligned} ECR(s,\epsilon) &= E(CR(s,T)|s,\epsilon) \\ &= \sum_{t \in \mathcal{T}(s)} P(T = t|s,\epsilon) CR(s,t) \qquad (2) \end{aligned}$$

The task in the troubleshooting problem is to find a successful troubleshooting strategy s^* that is optimal in the sense that it minimizes the expected cost of repair given the current evidence ϵ:

$$s^* = \arg\min_s ECR(s,\epsilon) \qquad (3)$$

3 Case Study — A Fuel Injection System

We will use an existing fuel injection system of a truck as a case study when evaluating the here proposed algorithms. This is a motivating and inspiring example from the real world since it is particularly hard for the on-board diagnosis system to isolate component faults.

The fuel injection system uses a high injection pressure to inject diesel in the cylinders of the engine. For many mechanical faults on this system, the only symptom is a loss in injection pressure. These faults can only be located by manual testing. A picture of the fuel injection system is shown in Figure 2.

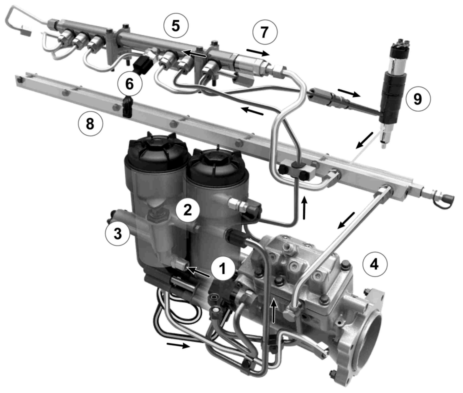

Figure 2: Extreme High Pressure Fuel Injection System.

The system is modeled with 17 components (pipes, pumps, filters, and valves) of which 7 are observable, i.e. a single observing action can unambiguously determine if the component is faulty. In total there are 30 binary observations that can be made (e.g. cylinder balancing, visible leakage, and air in fuel test). 21 of them can be accessed by observing actions. The remaining 9 observations are generated by the on-board diagnosis system of the truck and their values are given prior to the troubleshooting.

The dependencies between component faults and observations are modeled with a Bayesian network. Details on how this information was retrieved can be found in [Mossberg, 2007]. Apart from the assumption that at most one component can be faulty at the same time there are no causal dependencies between the faults. Observations are dependent on current component faults and on themselves in the sense that an observing action will always yield the same result when repeated unless any of the faults, that the observation is causally dependent on, is repaired.

The costs of the actions are heterogeneous, but in general the repair actions are more expensive than observing actions. The function control is the most expensive observing action since it requires a test drive of the vehicle.

4 Approximate and Exact Solutions to the Troubleshooting Problem in the Literature

In this section we will describe some different approaches to the troubleshooting problem. There are the greedy approaches mainly based on work by [Heckerman *et al.*, 1995] and [Langseth and Jensen, 2002] where time efficient algorithms are used to find approximate solutions to the troubleshooting problem. There are also various search based methods where more accurate solutions are found by searching at a higher cost in time.

4.1 Solving the Troubleshooting Problem Using Greedy Algorithms

In [Heckerman *et al.*, 1995] a special case of the troubleshooting problem is solved optimally in linear time using a greedy algorithm. We will take some time to describe this approach since we will compare our approach with this one in Section 7.3. The problem formulation of the basic troubleshooting problem is extended with the following assumptions:

- There can only be at most one faulty component.
- Immediately following any component repair a function control must be made.
- When the troubleshooting starts the problem-defining node is observed to be indicating, i.e. we know that exactly one component is faulty.
- Some components are observable, i.e. an observing action can be made that unambiguously determines if the component is functioning or not.
- No other observing actions are available.

Under these assumptions it is proved that an optimal troubleshooting strategy that minimizes the expected cost of repair can be obtained by always performing the action corresponding to the component with the highest *efficiency* first. The efficiency of an observable component is defined as the ratio of the probability that the component is faulty and the cost of observing it. For unobservable components the efficiency is defined as the ratio of the probability and the sum of the costs of repairing the component and making the function control.

Let $p_i = P(\text{component } i \text{ is faulty}|\epsilon)$. Then, using (2), the expected cost of repair using the strategy based on efficiencies s_{eff} on a system with n components is

$$ECR(s_{\text{eff}}, \epsilon) = \sum_{i=1}^{n} p_i \left(\left(\sum_{j=1}^{i} c_j^o \right) + c_i^r + c^{fc} \right) \quad (4)$$

where c_i^o is the cost of observing component i, c_i^r is the cost of repairing component i and c^{fc} is the cost of making the function control. An unobservable component is "observed" by first repairing it and then performing a function control. For these components c_i^o is the cost of repairing the component and making the function control and $c_i^r = -c^{fc}$ so that c_i^r and c^{fc} cancel each other in (4).

If any of the assumptions above are relaxed s_{eff} is no longer guaranteed to be optimal. However, near optimal troubleshooting strategies can be found by extending the strategy based on efficiencies with a two step look-ahead algorithm when more general observing actions also are available [Langseth and Jensen, 2002]. The two step look-ahead algorithm works by only allowing general observing actions to be either performed immediately in this time step or in the next.

Greedy algorithms can be constructed to solve the troubleshooting problem in other ways than using efficiencies. For example, in [de Kleer and Williams, 1987; Gillblad *et al.*, 2006] the next action to be performed is chosen as the one that maximizes the information gain. However, in empirical tests troubleshooting printers, the two step look-ahead algorithm is shown to find near-optimal solutions with an error of only a few percent [Vomlelová and Vomlel, 2000; Langseth and Jensen, 2002].

4.2 Solving the Troubleshooting Problem by Searching

Solving the troubleshooting problem optimally is a planning problem involving uncertainty and feedback. These types of problems can in a natural way be formulated as AND/OR graphs in which an optimal solution can be found by searching [Ghallab *et al.*, 2004; Bonet and Geffner, 2000]. AND/OR graphs are often used when solving the troubleshooting problem by searching. In [Vomlelová and Vomlel, 2000] they are used to describe and optimally solve small troubleshooting problems and in [Raghavan *et al.*, 1999; Olive *et al.*, 2003] they are used when finding optimal and near-optimal solutions to the *test sequencing problem.* The test sequencing problem is similar to the troubleshooting problem, where the goal is to isolate the fault by performing tests to progressively gain more information.

AND/OR Graphs

An AND/OR graph can be represented as a labeled directed hypergraph with a single root node [Nilsson, 1980]. The edges connect one parent node with one or more successor nodes. An edge connecting a parent node to k successor nodes is called a *k-connector*. AND/OR graphs are sometimes represented as regular directed graphs where the nodes of the hypergraph are called *OR nodes* and k-connectors with $k \geq 2$ are replaced by nodes called *AND nodes* with one incoming edge and k outgoing edges. An example of an AND/OR graph represented by a hypergraph is shown in Figure 3.

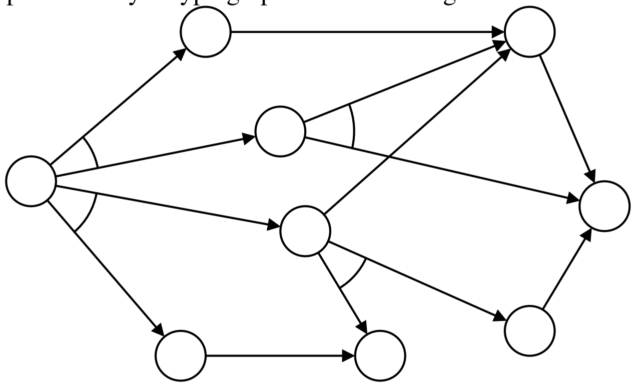

Figure 3: An AND/OR graph represented as a hypergraph. 2-connectors are shown as arrows joined with arcs.

When the AND/OR graph is used to describe the troubleshooting problem, each node represents a decision point where each outgoing connector represents an action that can be chosen to be performed. When a repair action is performed another decision point is reached. The nodes representing these decision points are connected by a 1-connector labeled with the repair action. When an observing action with k possible outcomes is performed, different decision points will be reached depending on the outcome. These nodes are connected by a k-connector labeled with the observing action.

A subgraph s of an AND/OR graph $\mathcal{G}$ is a *solution* of $\mathcal{G}$ corresponding to a troubleshooting strategy if the following conditions are true [Vomlelová and Vomlel, 2000]:

- the root of s is also the root of $\mathcal{G}$.

- if n is a non-terminal node in s, then *exactly one* outgoing connector from n also belong to s.
- if c is a connector in s, then *all* successor nodes of c also belong to s.
- all terminal nodes in s are leaf nodes in $\mathcal{G}$.

Since a solution corresponds to a troubleshooting strategy the cost of a solution can be defined by (2).

Algorithms for Finding Solutions in AND/OR Graphs

Describing the troubleshooting problem like this allows us to use existing algorithms used for finding solutions in AND/OR graphs such as *Value Iteration* [Bertsekas, 1995] and *AO** [Martelli and Montanari, 1978; Nilsson, 1980] to find optimal troubleshooting strategies.

Value Iteration is a general algorithm often used for finding optimal strategies in Markov Decision Processes. In [Cassandra *et al.*, 1998] Value Iteration is adapted to solve problems formulated as Partially Observable Markov Decision Processes which is a problem formulation similar to AND/OR graphs. In Value Iteration, estimates of the optimal solution are updated sequentially for every possible state of the problem until convergence. An advantage with this algorithm is that once a solution is found the optimal troubleshooting strategy is also found for every possible initial state. However, since an estimate needs to be assigned to every possible state Value Iteration becomes inefficient when the state space is large.

AO* is a search algorithm that makes use of a heuristic function to focus the search when finding optimal solutions in acyclic AND/OR graphs. The heuristic is used to estimate cost-to-go in each leaf node, i.e. the cost of an optimal solution of the subproblem rooted in the leaf node. When good heuristics are available AO* has been shown to perform better than Value Iteration for solving AND/OR graphs [Bonet and Geffner, 2005]. The AO* algorithms expand the AND/OR graph node by node and keep track of the currently best partial solution given heuristic estimates of the cost-to-go in each leaf. This gives AO* the advantage that it can be stopped prematurely returning a suboptimal partial solution.

A disadvantage with AO* is that it cannot handle cycles and that it is memory intensive. However, there are variants of AO* that handle cyclic AND/OR graphs [Hansen and Zilberstein, 2001], but it turns out that, for our problem, cycles in the AND/OR graph can be prevented by forbidding certain actions without loss of optimality. Also, there are memory bounded variants of AO* [Chakrabarti *et al.*, 1989] so that the memory usage can be controlled by trading space with time.

5 Using AO* to Solve the Troubleshooting Problem

In this section we will describe how AO* is implemented to solve troubleshooting problem such as the one described in Section 3. First we will give a brief overview of how the AO* algorithm works as described in [Nilsson, 1980]. Then we show how we do the state representation of the troubleshooting problem and how we treat this in our implementation. We will also describe how we use AO* to search with limited time.

5.1 Overview of the AO* Algorithm

Let $\mathcal{G}$ be the implicit AND/OR graph of the troubleshooting problem and let $\mathcal{G}'$ be the explicit subgraph of $\mathcal{G}$ consisting of the nodes and connectors that have been explored by the algorithm. Every node n in $\mathcal{G}'$ are associated with a *cost* q_n and labeled with a *state* $S \in \mathcal{S}$ that describes the system at a given moment where $\mathcal{S}$ is the state space. The cost of a leaf node n in $\mathcal{G}'$ that is not a leaf in $\mathcal{G}$ is given by a heuristic function $h : \mathcal{S} \mapsto \mathbb{R}$. The *best partial solution* is an arbitrarily chosen solution of $\mathcal{G}'$ that minimizes (2). Let this solution be denoted $\hat{s}$.

The algorithm starts the search with $\hat{s}$ consisting solely of the root node of $\mathcal{G}$. Until $\hat{s}$ is also a solution of $\mathcal{G}$, the node with the highest cost of the terminal nodes in $\hat{s}$ that are not leafs in $\mathcal{G}$ is expanded. Whenever a node is expanded, the currently best solution $\hat{s}$ is updated. The algorithm terminates once a solution of $\mathcal{G}$ is found. If the heuristic function is *admissible*, i.e. it never over-estimates the true cost-to-go, the solution corresponds to the troubleshooting strategy that minimizes (2).

5.2 State Representation

The state that we label each node with needs to contain the information necessary to describe our knowledge of the system at that point in the troubleshooting process. If we assume the Markov property, we can describe our knowledge with a *belief state* in accordance with [Russell and Norvig, 2003].

Let $f_{i,k} : i \geq 1$ be the event that component i is faulty at time k and let $f_{0,k}$ be the event that the system has no faults at time k. Let $e_{1:k}$ be the accumulated evidence from at time k. If no new information is gained, the probability that component i is faulty remains the same at time $k+1$:

$$P(f_{i,k+1}|e_{1:k}) = P(f_{i,k}|e_{1:k}) \tag{5}$$

Let us now assume that observations are only dependent on current component faults. Then:

$$P(e_k|f_{i,1:k}, e_{1:k-1}) = P(e_k|f_{i,k}) \tag{6}$$

where e_k is an observation made at time k.

If we know the probability distributions of all faulty components given all accumulated evidence at time $k-1$, we can calculate the probability distributions of all faulty components given the accumulated evidence at time k.

$$\begin{aligned} P(f_{i,k}|e_{1:k}) &= P(f_{i,k}|e_k, e_{1:k-1}) \\ &\propto P(e_k|f_{i,k}, e_{1:k-1})P(f_{i,k}|e_{1:k-1}) \\ &\overset{(5)(6)}{\propto} P(e_k|f_{i,k})P(f_{i,k-1}|e_{1:k-1}) \end{aligned} \tag{7}$$

This probability distribution is referred to as the belief state.

Definition 1. (Belief state) The *belief state* is a vector b_k containing the probability distribution over component faults given the accumulated evidence $e_{1:k}$ at time k. Each element $b_k(i)$ is

$$b_k(i) = P(f_{i,k}|e_{1:k}) \tag{8}$$

The belief state b_0 at time 0 is the à priori probability distribution over component faults. □

When an observing action is performed the evidence e_k is gained and we can calculate a new belief state b_k from the previous belief state b_{k-1}. For each element $b_k(i)$ in b_k

$$\begin{aligned} b_k(i) &\overset{\text{Def. 1}}{=} P(f_{i,k}|e_{1:k}) \\ &\overset{(7)}{\propto} P(e_k|f_{i,k})P(f_{i,k-1}|e_{1:k-1}) \\ &\propto P(e_k|f_{i,k})b_{k-1}(i) \end{aligned} \tag{9}$$

Since we assume single faults, when a repair action is performed we increase the probability that the system is free of faults with the probability that the repaired component was faulty and then we set that probability to zero. After component j is repaired, for each element $b_k(i)$ in b_k

$$b_k(i) = \begin{cases} b_{k-1}(0) + b_{k-1}(j) & \text{if } i = 0 \\ 0 & \text{if } i = j \\ b_{k-1}(i) & \text{otherwise} \end{cases} \tag{10}$$

5.3 Repeated Observing Actions

For the fuel injection system described in Section 3 we said that an observing action that is repeated always yields the same result unless any of the faults, that the observation is causally dependent on, is repaired. This means that no new information can be gained by that observing action. Even though this violates the Markov property, we do not have to change the probabilistic model if we add a small exception to (9).

We propose to keep track of all *recently made observing actions* at time k in a set $\mathcal{O}_k$. When a component is repaired we remove, from the same set, all observing actions that observes a node that is causally dependent on this component fault. Let $\mathcal{A}_i$ be the set of all observing actions observing nodes that are causally dependent on the component fault i. Then if component i is repaired the set of all recently made observations is updated such as

$$\mathcal{O}_k = \mathcal{O}_{k-1} \setminus \mathcal{A}_i \tag{11}$$

and when an observing action o is performed it is updated such as

$$\mathcal{O}_k = \mathcal{O}_{k-1} \cup \{o\} \tag{12}$$

Let e_k be the evidence gained from an observing action o. Then instead of using (9), after o is performed we update the belief state such that for each element $b_k(i)$ in b_k

$$b_k(i) = \begin{cases} b_{k-1}(i) & \text{if } o \in \mathcal{O}_{k-1} \\ P(e_k|f_{i,k})b_{k-1}(i) & \text{otherwise} \end{cases} \tag{13}$$

In order to describe the state of the system at a given moment we need to have a state that includes both the belief state and the set of recently made observations.

Definition 2. (State) A *state* S is a tuple containing a belief state b and a set of recently made observing actions $\mathcal{O}$:

$$S = \langle b, \mathcal{O} \rangle \tag{14}$$

□

A *goal state* is a state in which the probability that no component is faulty is one. Any node containing a goal state is a leaf in $\mathcal{G}$.

We will use the notation $S_n = \langle b_n \mathcal{O}_n \rangle = \langle b_k \mathcal{O}_k \rangle$ for the state description of a node n at time k meaning that the parent node of n is labeled with the state $S_{k-1} = \langle b_{k-1} \mathcal{O}_{k-1} \rangle$.

5.4 Expanding Nodes

When the algorithm expands a node, given the state a limited amount of actions can be performed. A repair action yields a 1-connector connected to a node labeled with a state created from the previous state using (10) and (11). If no node labeled with that state already exists in the explicit graph $\mathcal{G}'$, a new node is created. A binary observing action yields a 2-connector connected to nodes labeled with states created using (13) and (12).

When expanding, we will only consider actions that bring the system closer to the goal of repairing the system, i.e. repair actions that repair component faults with a probability greater than zero and observing actions from which new information can be gained. These actions are said to be *applicable actions*.

Definition 3. (Applicable Action) Let $\mathcal{B}'$ be the set of resulting belief states when action a is performed on the state $S = \langle b, \mathcal{O} \rangle$. An action a is *applicable* in S if there exists $b' \in \mathcal{B}'$ such that $b' \neq b$. □

Remark. When a binary observing action is performed, $\mathcal{B}'$ consists of two belief states, one for each possible outcome of the observation. When a repair action is performed $\mathcal{B}'$ consists a single belief state.

5.5 Updating the Best Partial Solution

Each connector is associated with an action a and a probability p_m for each successor node m. For a 2-connector p_m is the probability of having the corresponding outcome e_k given the accumulated evidence $e_{1:k-1}$:

$$\begin{aligned} p_m &= P(e_k|e_{1:k-1}) \\ &= \sum_i P(e_k|f_{i,k}, e_{1:k-1})P(f_{i,k}|e_{1:k-1}) \\ &= \sum_i P(e_k|f_{i,k})b_{k-1}(i) \end{aligned} \tag{15}$$

For a 1-connector $p_m = 1$ since there is only one outcome. The cost of a connector c, $k(c)$, is a function of the action cost c_a and the probabilities p_m for each node m in the set of successor nodes $succ(c)$.

$$k(c) = c_a + \sum_{m \in succ(c)} p_m q(m) \tag{16}$$

When the algorithm has expanded a node n, the cost q_n is updated such that

$$q_n = \min_{c \in outg(n)} k(c) \tag{17}$$

where $outg(n)$ is the set of all outgoing connectors from n. The connector c that minimizes (17) is included in a partial solution of the subproblem rooted in n. Until a partial solution is found for the root node, the same procedure is done for all predecessors of n in $\hat{s}$. When this is done, the partial solution for the root node is now the new best partial solution.

Recall that in the problem formulation in Section 2 we said that the terminal node in a troubleshooting strategy is successful only if the previous action is a successful function control. This means that even if we believe that the system is free of faults we still have to make the function control to complete the troubleshooting. Let this action be a with the cost c^{fc}.

Then the cost q_n of a node n that is terminal in the explicit graph $\mathcal{G}$ labeled with the goal state $S = \langle b, \mathcal{O} \rangle$ is

$$q_n = \begin{cases} 0 & \text{if } a \in \mathcal{O} \\ c^{fc} & \text{otherwise} \end{cases} \tag{18}$$

Note that if have the function control $a \in \mathcal{O}$ in the goal state, this action must have been the most recently performed action.

A terminal n node in $\mathcal{G}'$ labeled with a state S_n that is not a goal state cannot be a terminal node in $\mathcal{G}$. The cost of this node is given by a heuristic function h.

$$q_n = h(S_n) \tag{19}$$

5.6 Searching with Limited Time

Finding an optimal solution to a large AND/OR graph cannot be done efficiently even with a fairly good heuristic. If we want the algorithm to come up with a solution while the user is waiting, it must finish in reasonable time. Therefore, after a certain time T we will forbid the expansion of more nodes. Instead, the cost of the nodes, that should have been expanded, is set to a certain *cut-off cost* and the best partial solution $\hat{s}$ is returned. This makes our search incomplete and the solution can no longer be guaranteed to be optimal. In [Sadikov and Bratko, 2006] it is argued that an optimistic heuristic function can degrade the result when used with incomplete heuristic search methods. Therefore, we propose to use a cut-off cost that gives us a pessimistic estimate of the optimal cost-to-go. The troubleshooting strategy based on efficiencies s_{eff} described in Section 4.1 has this property. Since s_{eff} assumes that the system is faulty, the cost of making the function control c^{fc} is added, if there is a non-zero probability that the system is free of faults. If the system has a fault, the expected cost is calculated using (4). For a node n with the state $S_n = \langle b_n, \mathcal{O}_n \rangle$, let the integers $a_1, a_2, \dots, a_m$ be the indexes of components ordered by efficiencies in descending order. When n is cut off the cost q_n of that node is set to be

$$q_n = \lceil b_n(0) \rceil c^{fc} + (1 - b_n(0)) \sum_{i=1}^{m} \left(b_n(a_i) \left(c^r_{a_i} + c^{fc} + \sum_{j=1}^{i} c^o_{a_j} \right) \right) \tag{20}$$

where c^o_j and c^r_j are action costs of observing respectively repairing component j.

6 The Heuristic Function

The AO* algorithm will find the optimal solution provided an admissible heuristic function, i.e. a function that never overestimates the optimal cost-to-go from a node. As concluded in the previous section, when searching with limited time, the solution is no longer guaranteed to be optimal. Therefore, it is not necessary that the heuristic function is admissible. The aim of the new heuristic function is to minimize the relative error in the estimated cost-to-go.

In [Vomlelová and Vomlel, 2000] an admissible heuristic function is used together with AO* for solving the troubleshooting problem in the domain of home electronics. It is derived from a relaxation of the troubleshooting problem where we can make a "perfect" observing action that points out the true underlying component fault at no cost. The optimal cost of repair of the relaxed problem is easily calculated. It is the cost of repairing the faulty component weighted with its probability plus the cost of a final function control. It is well known that admissible heuristics can be acquired by solving a relaxation of the problem optimally [Russell and Norvig, 2003]. Let $h_1 : \mathcal{S} \mapsto \mathbb{R}$ be this heuristic. For a system with n components and the state $S = \langle b, \mathcal{O} \rangle$, i.e.

$$h_1(S) = c^{fc} + \sum_{i=1}^{n} b(i) c^r_i \tag{21}$$

where c^{fc} is the cost of the function control and c^r_i is the cost of repairing component i.

A problem with this heuristic is that it only considers the cost of the repair actions. This means that, if many observing actions are included in the optimal troubleshooting strategy, h_1 returns a value much too low. This is the case for the fuel injection system described in Section 3. Therefore we need a stronger heuristic.

A feature of the troubleshooting problem that is ignored by h_1, is our uncertainty of which component is faulty. The *entropy* of the probability distribution can be used as a measure of this uncertainty [Gray, 1990]. Let $S = \langle b, \mathcal{O} \rangle$ be the state of a node. Then the entropy of the probability distribution in b, $H(b)$, is given by

$$H(b) = -\sum_{i=0}^{n} b(i) \log b(i) \tag{22}$$

where n is the number of components.

In experiments on the fuel injection system we have measured the optimal cost-to-go q^*_n and the state $S_n = \langle b_n, \mathcal{O}_n \rangle$ of every node n for which the optimal solution could be found using the h_1 heuristic. In Figure 4 we have plotted $q^*_n - h_1(S_n)$ against $H(b_n)$. As $H(b_n)$ grows a linear trend in the difference $q^*_n - h_1(S_n)$ is visible. This is caused by the extra observing actions needed isolate component faults.

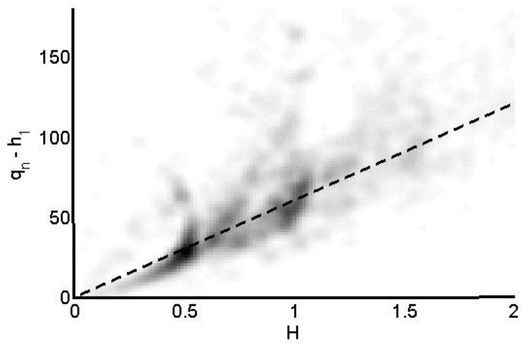

Figure 4: Plot of $q^*_n - h_1(S_n)$ against $H(S_n)$ of 4250 measured nodes. Darker areas indicate a higher density of measurements.

The heuristic function that we propose exploits this linear trend to get a better approximation of the estimated cost-to-go. The linear trend is modeled by a parameter $\hat{c}_H$ that describes the estimated cost of reducing entropy. We will call this the *entropy cost*. The new heuristic function $h_2 : \mathcal{S} \mapsto \mathbb{R}$ is defined as

$$h_2(S) = h_1(S) + \hat{c}_H H(S) \tag{23}$$

As with h_1, this heuristic does not require any expensive computation but it is not admissible and it requires a set of training data from the problem domain so that a value of $\hat{c}_H$ can be assigned.

We want to keep the relative error $\varepsilon(n)$ of the optimal cost-to-go q_n^* and the heuristic value $h_2(S_n)$ minimal for every node n, i.e.

$$\varepsilon(n) = \frac{|q_n^* - h_2(S_n)|}{q_n^*} \tag{24}$$

We do this by fitting the parameter $\hat{c}_H$ in (23) to data from a training set of simpler problems in which q_n^* is known using linear regression. For the fuel injection system in our case study this value was 60.3 and it is indicated by a dashed line in Figure 4.

7 Empirical Evaluation

To evaluate the new heuristic we will study how the relative error and the size of the search graph grows with problem size. We will also study how the new heuristic affects the performance of troubleshooting when we have limited time.

7.1 Relative Error

We can only measure the relative error on problems that can be solved optimally. These problem instances are obtained by making a series of random actions on the case study until the remaining problem is small enough to be solved using AO* with the admissible heuristic h_1. As a measurement of problem size we use the number of applicable actions available.

We will use as a reference a heuristic that is based on an analogy to the Huffman coding problem which is used with AO^* to solve the test sequencing problem [Raghavan *et al.*, 1999]. Let $c_1, c_2, \ldots$ be the costs of the observing actions that are applicable in the state S ordered such that $c_1 \leq c_2 \leq \ldots$. Then the heuristic function h_3 is given by

$$h_3(S) = \sum_{i=1}^{\lfloor H(S) \rfloor} c_i + \big(H(S) - \lfloor H(S) \rfloor\big) c_{\lfloor H(S) \rfloor + 1} \tag{25}$$

Since the test sequencing problem does not involve repair actions we will combine h_1 and h_3 to get a more fair comparison. We will call the resulting heuristic h_4:

$$h_4(S) = h_1(S) + h_3(S) \tag{26}$$

Figure 5 shows a comparison of the relative errors of h_1, h_2, and h_4 on a different data set than the one used to calculate $\hat{c}_H$ with a problem size varying from 3 to 12.

Since the parameter $\hat{c}_H$ is designed to minimize the relative error, this value is the lowest for the h_2 heuristic. The relative error is approximately constant for all problem sizes in the experiment which indicates that h_2 is equally strong also for larger problem sizes.

7.2 Size of the Search Graph

To show the difference in the ability to focus the search, we have measured the number of created nodes $\mathcal{G}'$ and the relative error from optimum δ when the algorithm is run with different heuristics on solvable problems with growing size. Let ECR_i be the expected cost of the solution found using the heuristic

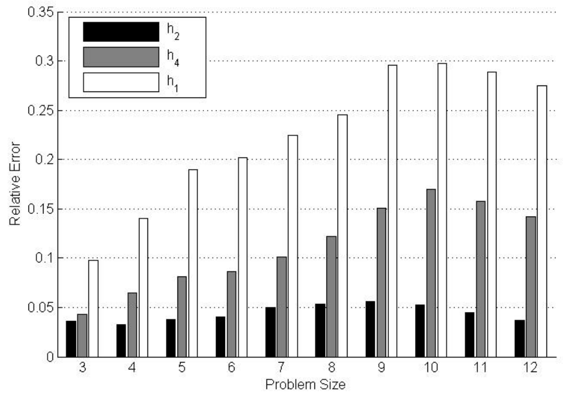

Figure 5: Mean relative error of h_1, h_2, and h_4 measured in 3596 solved nodes.

h_i and let ECR^* be the expected cost of the optimal solution. Then the relative error of the heuristic h_i is

$$\delta(h_i) = \frac{|ECR^* - ECR_i|}{ECR^*} \tag{27}$$

Since h_1 is admissible, $ECR_1 = ECR^*$. The sizes of the search graphs for h_1, h_2, and h_4 are shown in Figure 6. The relative error δ was at all times below 0.01 for both h_2 and h_4.

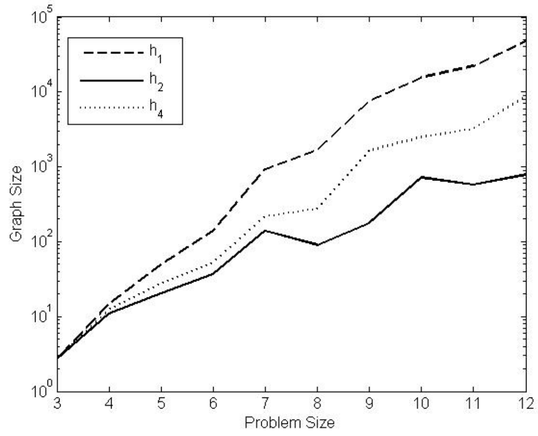

Figure 6: Mean sizes of the search graphs for 229 problems of varying size searched using the heuristics h_1, h_2, and h_4.

The exponential growth of the search graph is the least for the h_2 heuristic. This allows for finding near-optimal solutions of larger problems.

7.3 Troubleshooting with Limited Time

When troubleshooting with limited time we will only have a partial solution. The first action in this solution is performed on the system and a new partial solution is calculated. In this experiment, we will compare our version of AO* with limited time using h_1 and h_2 with the greedy two step look-ahead troubleshooting algorithm presented in [Langseth and Jensen, 2002].

We will let the troubleshooting algorithms troubleshoot the fuel injection system with predefined hidden faults. The system will respond to actions according to the probabilistic model. The troubleshooting stops when the correct component is repaired and a confirming functional control is made.

Each algorithm is allowed 10 seconds to decide the next action to perform. In 10 seconds the search algorithm expands approximately 30000 nodes. The mean cost of repair is measured for 100 randomly generated problem instances. The results are shown in Table 1.

Time limited AO* using h_1	556.95
Time limited AO* using h_2	476.91
Time limited AO* using h_4	518.52
Greedy two step look-ahead	618.57

Table 1: Comparison of the mean cost of repair for the troubleshooting algorithms.

The greedy two step search found its solutions within milliseconds, but since a function control always is required after each repair action the costs became higher. The time limited search algorithm is not constrained by this and thereby the costs were less. The h_2 measured the lowest costs. This is mainly due to that larger parts of the search graph could be explored during the 10 seconds.

8 Conclusions

We have shown how the AO* algorithm can be used to solve the troubleshooting problem on a case study from the automotive domain. We have shown that it can be used to recommend troubleshooting actions within a fixed time limit. We have presented a new entropy based heuristic. In the empirical evaluations we have shown that, by using this heuristic, the troubleshooting costs can be reduced compared to when using the greedy two-step look-ahead algorithm or any of the heuristics h_1 and h_4.

Acknowledgments

For fruitful discussions, we would like to thank our colleagues at the Dept. of Computer and Information Science at Linköping University, in particular Jonas Kvarnström and Per Nyblom. We would also like to thank our coworkers at Scania for all the support. This research has been funded by VINNOVA V-ICT and Scania CV AB.

References

[Bertsekas, 1995] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Bonet and Geffner, 2000] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Artificial Intelligence Planning Systems*, pages 52–61. 2000.

[Bonet and Geffner, 2005] Blai Bonet and Hector Geffner. An Algorithm Better than AO*? In *AAAI*, pages 1343–1348. AAAI Press / The MIT Press, 2005.

[Cassandra *et al.*, 1998] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, pages 99–134, 1998.

[Chakrabarti *et al.*, 1989] P.P. Chakrabarti, S. Ghose, A. Acharya, and S. C. de Sarkar. Heuristic search in restricted memory. *Artificial Intelligence*, 41(2):197–222, 1989.

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

[Faure, 2001] P.P. Faure. *An Interval Model-Based Approach for Optimal Diagnosis Tree Generation: Application to the Automotive Domain*. PhD thesis, LAAS, 2001.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning*. Morgan Kaufmann, San Francisco, 2004.

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

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

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

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

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

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

[Martelli and Montanari, 1978] Alberto Martelli and Ugo Montanari. Optimizing decision trees through heuristically guided search. *Commun. ACM*, 21(12):1025–1039, 1978.

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

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

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

[Raghavan *et al.*, 1999] Vijay Raghavan, M. Shakeri, and Krishna R. Pattipati. Optimal and near-optimal test sequencing algorithms with realistic test models. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 29(1):11–26, 1999.

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

[Sadikov and Bratko, 2006] Aleksander Sadikov and Ivan Bratko. Pessimistic Heuristics Beat Optimistic Ones in Real-Time Search. In *ECAI*, pages 148–152. IOS Press, 2006.

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