Soft Computing 7 (2003) 357–368 © Springer-Verlag 2003
DOI 10.1007/s00500-002-0224-4

Troubleshooting: NP-hardness and solution methods

M. Vomlelová, J. Vomlel*

Abstract Troubleshooting is one of the areas where Bayesian networks are successfully applied [9]. In this paper we show that the generally defined troubleshooting task is NP-hard. We propose a heuristic function that exploits the conditional independence of all actions and questions given the fault of the device. It can be used as a lower bound of the expected cost of repair in heuristic algorithms searching an optimal troubleshooting strategy. In the paper we describe two such algorithms: the depth first search algorithm with pruning and the AO* algorithm.

Keywords Decision-theoretic troubleshooting, Computational complexity, Bayesian networks

1
Introduction

We model the troubleshooting problem with a Bayesian network encoding relations among the following variables: *faults* of the device $F \in \mathscr{F}$, *actions* $A \in \mathscr{A}$ – troubleshooting steps that may fix the problem, and *questions* $Q \in \mathscr{Q}$ – troubleshooting steps that help to identify the fault. Every action and question has a cost assigned, c_A denotes the cost of an action A and c_Q the cost of a question Q. When there is no risk of confusion we will abbreviate c_{A_i} to c_i. Before we introduce formal definitions we will start with a simple example.

1.1
Light print example

Suppose a printer prints a page that is too light. The possible printer faults in the case of a light print problem are listed in Table 1. Please note, that realistic model of light print designed by the domain experts invokes 22 faults. Let us consider a simplified model with four faults only.

There are several actions that may fix these faults (see Table 2). For example, the action *Try another toner* fixes the *Distribution problem* and *Defective toner* with the probability 0.9 (i.e. $p(A_2 = yes \mid F_1) = p(A_2 = yes \mid F_2) = 0.9$), but it does not fix *Wrong driver setting* at all, i.e. $p(A_2 = yes \mid F_4) = 0$.

Generally, for every action an expert provides us with a table $p(A_i = yes \mid F_j)$. The conditional probability table of $p(A_2 = yes \mid F_j)$ is defined in Table 3.

During the troubleshooting session it is often advisable to ask the user to answer some questions. The answers may help fix the problem faster by identifying the device fault. For instance if the answer to question Q_1: *Is the configuration page printed light*? is negative then the faults: *Distribution problem* and *Defective toner* are eliminated and the remaining faults are *Corrupt data flow* and *Wrong driver setting*. Generally, we have a table $p(Q_i = yes \mid F_j)$ for every question Q_i. See Table 3 for the definition of the conditional probability table of $p(Q_1 = yes \mid F_j)$. Actions and questions are *conditionally independent* if for each $A_i \in \mathscr{A}, Q_k \in \mathscr{Q}$

$$p(A_i|\mathscr{F}) = p(A_i|\mathscr{F},\mathscr{V}) \text{ for any } \mathscr{V} \subseteq (\mathscr{A} \cup \mathscr{Q}) \setminus \{A_i\}$$
$$p(Q_k|\mathscr{F}) = p(Q_k|\mathscr{F},\mathscr{U}) \text{ for any } \mathscr{U} \subseteq (\mathscr{A} \cup \mathscr{Q}) \setminus \{Q_k\} \ .$$

When there is only one fault causing a device malfunction at a time then it is often referred to as the *single fault assumption*. The Bayesian network in Fig. 1 reflects both assumptions.

There are many possible troubleshooting strategies. For example, first try action A_3 : *Cycle power* and then stop troubleshooting. Another strategy is: A_3 : *Cycle power* and if this is unsuccessful, A_2 : *Try another toner*. A third strategy may require the user first to answer question Q : *Is the configuration page printed light*? If the answer is *yes*, then A_1 : *Remove shake and reseat toner*. If the answer is *no*, then try A_3 : *Cycle power*. One criteria for comparing different strategies is the expected cost of repair (ECR). Table 4 shows the calculations of ECR for the three examples presented above, where c_{A_i} is the cost of action A_i and c_{CS} is a penalty for not solving the problem. Please note that real life troubleshooting strategies contain substantially more actions and questions than those considered here.

1.2
Troubleshooting task specification

If only actions are considered, then every troubleshooting strategy can be described as a sequence of actions that are performed until the problem is fixed. When questions are part of troubleshooting, then the solution of a trouble-

---

M. Vomlelová (✉), J. Vomlel
Research Unit of Decision Support Systems,
Department of Computer Science, Aalborg University,
Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark

* The authors were supported through grant #87.2 of National Centre for IT Research, Denmark and through grant MSMT VS96008 from the Ministry of Education, Youth and Sports of the Czech Republic.

We would like to thank Finn Verner Jensen for inspiring us to work on the discussed problem and for many valuable comments on this paper. We are grateful to Claus Skaanning for the detailed explanation of the BATS troubleshooter approach and to anonymous reviewers for helpful suggestions.

Table 1. Possible faults in the case of light print

Fault	$p(F_i)$
F_1: *Distribution problem*	0.4
F_2: *Defective toner*	0.3
F_3: *Corrupted dataflow*	0.2
F_4: *Wrong driver setting*	0.1

Table 2. Actions and one question in the light print example

Actions and questions	c_i
A_1: *Remove, shake and reseat toner*	5
A_2: *Try another toner*	15
A_3: *Cycle power*	1
Q_1: *Is the configuration page printed light?*	2

Table 3. Conditional probability tables

F_j	$p(A_2 = yes \mid F_j)$	F_j	$p(Q_1 = yes \mid F_j)$
F_1	0.9	F_1	1
F_2	0.9	F_2	1
F_3	0	F_3	0
F_4	0	F_4	0

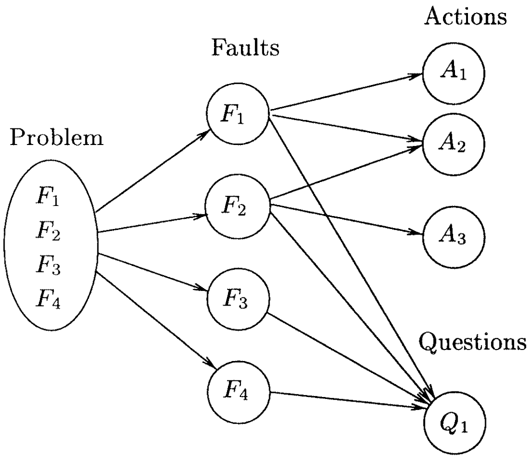

Fig. 1. Bayesian network model

Table 4. ECR calculations

Strategy	**ECR**
A_3	$c_{A_3} + p(A_3 = no) \cdot c_{CS}$
A_3, A_2	$c_{A_3} + p(A_3 = no) \cdot c_{A_2}$ $+ p(A_2 = no, A_3 = no) \cdot c_{CS}$
$Q\begin{cases} A_1 \\ A_3 \end{cases}$	$c_Q + p(Q = yes) \cdot c_{A_1}$ $+ p(A_1 = no, Q = yes) \cdot c_{CS}$ $+ p(Q = no) \cdot c_{A_3}$ $+ p(A_3 = no, Q = no) \cdot c_{CS}$

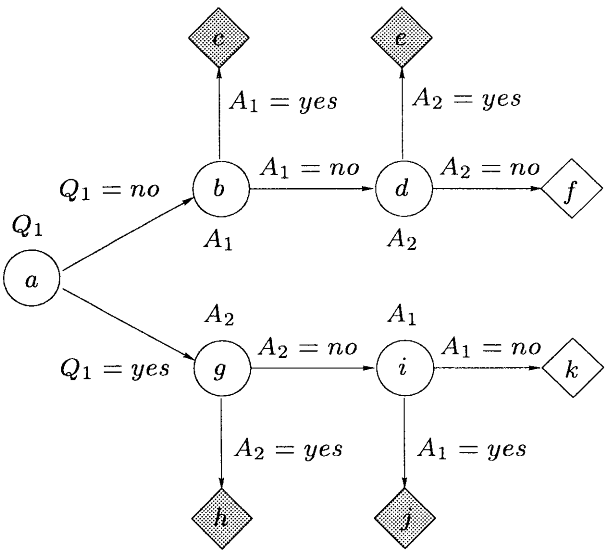

Fig. 2. Troubleshooting strategy

shooting task need not to be a sequence. For every answer to a question, a strategy may be to perform different troubleshooting steps (or the same steps in different order). Thus a troubleshooting strategy has generally the form of a directed tree where branching may occur after every question. Fig. 2 provides an example of such a troubleshooting strategy.

There are two types of nodes in the tree – *chance nodes* and *terminal nodes*. In Fig. 2 circles are used to denote chance nodes and the diamonds to denote terminal nodes. Each chance node n is labeled by the corresponding troubleshooting step (action or question) provided by function *step*(n). An example is *step*$(a) = Q_1$ in Fig. 2. Every edge coming out from a chance node is labeled by an outcome of the troubleshooting step corresponding to that node, *outcome*(*edge*) will denote the function that provides the *edge* labels.

A troubleshooting strategy terminates in *terminal nodes*. There are two ways for a troubleshooting strategy to terminate, either by fixing the problem or by giving up. Thereupon two types of *terminal nodes* are defined: (1) *Success terminal nodes* correspond to fixing the problem. (2) *Failure terminal nodes* correspond to giving up the troubleshooting. In Fig. 2 the success terminal nodes are shaded while failure terminal nodes are not.

Troubleshooting strategy is a labeled directed tree that describes the process of performing actions and questions until the process terminates. The set of all terminal nodes of a strategy $\mathbf{s}$ will be denoted by $\mathscr{L}(\mathbf{s})$. Please note that all leaves of a troubleshooting strategy are terminal nodes. The root node of all strategies will be denoted ϑ.

Let *path*(n_1, n_k) be sequence of edges $(n_i \rightarrow n_{i+1})_{i=1}^{k-1}$ constituting the path from node n_1 to node n_k in the troubleshooting strategy and

$$\mathbf{e}_n = \bigcup_{edge \in path(\vartheta, n)} outcome(edge)$$

define the evidence compiled through the performance of actions and questions in *path*(ϑ, n). Further let $p(\mathbf{e}_n \mid \mathbf{e}_m)$ for $\mathbf{e}_m \subseteq \mathbf{e}_n$ denote conditional probability of evidence $\mathbf{e}_n$ given evidence $\mathbf{e}_m$, i.e. the probability of getting to node n

from node m. Please note that since $\mathbf{e}_\vartheta = \emptyset$, the probability of getting to node n from the root is $p(\mathbf{e}_n)$. Finally let the total cost of actions and questions corresponding to a $path(n_1, n_k)$ be $t(n_1, n_k) = \sum_{\ell=1}^{k-1} c_{step(n_\ell)}$. As an example, in Fig. 2, the evidence corresponding to node b labeled by A_1 is $Q_1 = no$, the probability of getting there is $p(Q_1 = no)$, and the total cost of getting there is c_{Q_1}.

A penalty function $c(\mathbf{e}_\ell)$ applies for every terminal node ℓ. The penalty is defined to be zero if the problem is fixed, i.e. for all success terminal nodes. The penalty may be interpreted as a cost of calling service. If the penalty is constant for all failure terminal nodes then it will be denoted by c_{CS}. Next, we shall define the expected cost of repair of a troubleshooting strategy.

Definition 1 Expected cost of repair *(ECR) of a troubleshooting strategy* $\mathbf{s}$ *is defined as*

$$\mathrm{ECR}(\mathbf{s}) = \sum_{\ell \in \mathscr{L}(\mathbf{s})} p(\mathbf{e}_\ell) \cdot (\ t(\vartheta, \ell) + c(\mathbf{e}_\ell)\ ). \qquad (1)$$

Remark 1 If troubleshooting strategy $\mathbf{s}$ is a sequence of actions $A_1, A_2, \ldots, A_n$ and the penalty is constant c_{CS} then ECR of strategy $\mathbf{s}$ can be computed by

$$\begin{aligned} ECR(\mathbf{s}) &= p(A_1 = yes) \cdot c_1 \\ &\quad + p(A_1 = no, A_2 = yes) \cdot (c_1 + c_2) + \cdots \\ &\quad + p(A_1 = no, \ldots, A_{n-1} = no, A_n = yes) \cdot \sum_{i=1}^{n} c_{A_i} \\ &\quad + p(A_1 = no, \ldots, A_n = no) \cdot \left(\sum_{i=1}^{n} c_{A_i} + c_{CS} \right) \\ &= c_1 + \sum_{i=2}^{n} p(A_1 = no, \ldots, A_{i-1} = no) \cdot c_i \\ &\quad + p(A_1 = no, \ldots, A_n = no) \cdot c_{CS} \qquad (2) \end{aligned}$$

For a strategy $\mathbf{s}$ and a node m of this strategy, let symbol $\mathbf{s}_m$ denote a sub-strategy of $\mathbf{s}$ such that m is the root of $\mathbf{s}_m$ and all successors of m in $\mathbf{s}$ are also contained in $\mathbf{s}_m$. Please note that $\mathbf{s} = \mathbf{s}_\vartheta$.

It will be useful to have Definition 1 generalized so that the expected cost of repair of a troubleshooting sub-strategy $\mathbf{s}_m$ given an evidence $\mathbf{e}_m$ corresponding to root m of $\mathbf{s}_m$ is defined. It can be further extended to the case where an additional evidence $\mathbf{e}', \mathbf{e}' \cap \mathbf{e}_m = \emptyset$ is compiled. Please observe that for $m = \vartheta$ and $\mathbf{e}' = \emptyset$ we get Definition 1.

Definition 2 Let $\mathbf{e}', \mathbf{e}' \cap \mathbf{e}_m = \emptyset$ correspond to additional evidence. *Expected cost of repair* of a troubleshooting sub-strategy $\mathbf{s}_m$ given the evidence $\mathbf{e} = \mathbf{e}_m \cup \mathbf{e}'$ is defined as

$$\mathrm{ECR}(\mathbf{s}_m \mid \mathbf{e}) = \sum_{\ell \in \mathscr{L}(\mathbf{s}_m)} p(\mathbf{e}_\ell \mid \mathbf{e}) \cdot (\ t(m, \ell) + c(\mathbf{e}_\ell)\ ). \quad (3)$$

Remark 2 Definition 2 allows a troubleshooting step S to be part of $\mathbf{e}_\ell$ and $\mathbf{e}$ at the same time. However when searching a strategy minimizing ECR we can exclude such strategies. Assume $\{S = o_1\} \in \mathbf{e}_\ell$ and $\{S = o_2\} \in \mathbf{e}$. If $o_1 = o_2$ then $p(\mathbf{e}_\ell \mid \mathbf{e}) = p(\mathbf{e}_\ell \setminus \{S = o_1\} \mid \mathbf{e})$ else $p(\mathbf{e}_\ell \mid \mathbf{e}) = 0$. If $c_S > 0$ then no strategy $\mathbf{s}_m$ containing S can minimize $\mathrm{ECR}(\mathbf{s}_m \mid \mathbf{e})$ since it can be improved by a strategy with S excluded.

The following lemma provides directions for a recursive computation of ECR.

Lemma 1 *If* $\mathbf{e} \subseteq \mathbf{e}_m$, *then* $ECR(\mathbf{s}_m \mid \mathbf{e})$ *can be computed by the following recurrent formula:*

- *If m is a terminal node, then* $ECR(\mathbf{s}_m \mid \mathbf{e}) = c(\mathbf{e})$.
- *If m is a chance node corresponding to a troubleshooting step S (i.e. S is either a question or an action) with outcomes* $s_1, s_2, \ldots, s_r$ *and* $m_1, m_2, \ldots, m_r$ *are children of node m, then*

$$ECR(\mathbf{s}_m \mid \mathbf{e}) = c_S + \sum_{i=1}^{r} p(S = s_i \mid \mathbf{e}) \cdot ECR(\mathbf{s}_{m_i} \mid \mathbf{e} \cup S = s_i)\ . \qquad (4)$$

The lemma is simple to prove using induction over the tree structure of a troubleshooting strategy (starting from the leaves of strategy $\mathbf{s}$).

An optimal strategy given evidence $\mathbf{e}$ will be denoted by $\mathbf{s}^\star(\mathbf{e})$. Please note that the order in which the previous troubleshooting steps are performed (so that an evidence $\mathbf{e}$ is achieved) has no impact on the optimal strategy $\mathbf{s}^\star(\mathbf{e})$. We will abbreviate $ECR(\mathbf{s}^\star(\mathbf{e}) \mid \mathbf{e})$ to $ECR^\star(\mathbf{e})$. If $\mathbf{e} = \emptyset$, then we will simply write $\mathbf{s}^\star$.

Definition 3 The *troubleshooting task* is to find a troubleshooting strategy $\mathbf{s}^\star$ such that for all possible strategies s it holds that $ECR(\mathbf{s}^\star) \leq ECR(\mathbf{s})$.

1.3
Various setups of troubleshooting

It is often reasonable to assume that, if the device is malfunctioning, then there is only one fault in the device (*single fault assumption*) [3]. Otherwise the situation is referred to as *multiple faults*. When the faults are independent, i.e. $p_3(F_1, \ldots, F_{|\mathscr{F}|}) = \prod_{\mathscr{F}_i \in \mathscr{F}} p(\mathscr{F}_i)$, we speak about *independent faults* [10]. A solution of the *troubleshooting task* can be easily found in the case of *independent actions*, that is in the situations when (1) every action fixes just one fault and (2) all actions are pairwise independent.

In the case of *independent actions* with *single fault assumption* it suffices to order actions decreasingly according to the ratio $\frac{p(A_i = yes)}{c_{A_i}}$ (see [5]). In the case of *independent actions* with *independent faults* an ordering according to $\frac{p(A_i = yes)}{c_{A_i} \cdot (1 - p(A_i = yes))}$ leads to an optimal sequence [10].

The task becomes harder if some actions fix more than one fault. This case is referred to as *dependent actions*. In any case we assume that all actions and questions are pairwise independent given the faults. Please note that these assumptions can be expressed by different Bayesian network structures. The *single fault assumption* leads to a model with one single root node, the *problem node*, having all faults as its states (see Fig. 1), while the model of *independent faults* omits the problem node.

In the next section we shall analyze the complexity of troubleshooting. We will prove the NP-hardness of general troubleshooting with dependent actions under either the single fault assumption, or the independent faults assumption. If every action fixes at most two faults, we conjecture the troubleshooting is also NP-hard even if we show a polynomially solvable special case.

2
Complexity of troubleshooting

To prove that the troubleshooting with dependent actions is NP-hard we reduce the *exact cover by 3-sets* to troubleshooting.

Exact cover by 3-sets: We are given a family $\mathscr{S} = \{S_1, \ldots, S_n\}$ of subsets of a set U, such that $|U| = 3m$ for any integer m, and $|S_i| = 3$ for all i. We are asked if there are m sets in $\mathscr{S}$ that are disjoint and have U as their union. The proof of NP-completeness can be found in a number of works, including [6].

2.1
Reduction

Let us have an *exact cover by 3-sets* input $\langle \mathscr{S}, U \rangle$. We construct a troubleshooting task as follows:

- For each element i in U we construct a fault $\mathscr{F}_i$ in the troubleshooter.
- For each set $S_i \in \mathscr{S}$ we assign $\mathscr{F}_i$, a set of faults corresponding to the elements of S_i. We construct an action A_i that solves faults in $\mathscr{F}_i$ with probability 1 and others with probability 0.
- All faults are equally probable; the probability of any fault is $p(\mathscr{F}_i) = \frac{1}{3 \cdot m}$.
- All actions have cost 1.
- The cost of the Call Service action CS is high enough to ensure it is not used in any good strategy. We set it to $c_{CS} = 3 \cdot m \cdot (m+1)$. We will see later that this is high enough.

2.2
Single fault

We aim to prove that the *exact cover by 3-sets* exists if and only if there is a troubleshooting sequence $\mathbf{s}$ with $ECR(\mathbf{s}) \leq \frac{m+1}{2}$. We have to prove three lemmas.

Lemma 2 (Basic properties) *For any* r, $1 < r \leq 3m$, *and any* r *faults* $F_1, \ldots, F_r$, *the single fault assumption implies*

$$p(F_1 = yes \vee \ldots \vee F_r = yes) = \sum_{i=1}^{r} p(\mathscr{F}_i = yes) = \frac{r}{3m},$$

therefore
$p(F_1 = no, \ldots, F_r = no) = 1 - \sum_{i=1}^{r} p(\mathscr{F}_i = yes) = \frac{3m-r}{3m}$.
Since $p(A_i \mid F_j \in \mathscr{F}_i) = 1$ *we have*

$$p(A_1 = no, \ldots, A_r = no) = 1 - \sum_{F \in \bigcup_{i=1}^{r} \mathscr{F}_i} p(F = yes) = 1 - \frac{|\bigcup_{i=1}^{r} \mathscr{F}_i|}{3m} \quad . \tag{5}$$

The proof comes from the basic probability calculus.

Lemma 3 *If we have an exact cover by 3-sets* $V = \{S_{j_1}, \ldots, S_{j_m}\}$, *then the corresponding action sequence* $A_{j_1}, \ldots, A_{j_m}$ *(in any order) has*

$$ECR(A_{j_1}, \ldots, A_{j_m}) = \frac{m+1}{2} \quad .$$

Proof:
No two actions solve the same fault and each action solves 3 faults, therefore $\frac{|\cup_{i=1}^{r} \mathscr{F}_{j_i}|}{3m} = \frac{3r}{3m}$. Substituting this to (5) we get:

$$p(A_{j_1} = no, \ldots, A_{j_r} = no) = 1 - \frac{3r}{3m} = \frac{m-r}{m}$$

Now we calculate the *ECR* using formula 2 (the penalty does not apply since the sequence of actions is certain to solve the problem):

$$ECR(A_{j_1}, \ldots, A_{j_m}) = \sum_{i=1}^{m} p(A_{j_1} = no, \ldots, A_{j_{(i-1)}} = no) \cdot c_i \quad .$$

Please, recall that all actions have cost 1. Therefore:

$$ECR(A_{j_1}, \ldots, A_{j_m}) = \sum_{i=1}^{m} \left(\frac{m-(i-1)}{m} \right) \cdot 1 = \frac{(m+1) \cdot m}{2 \cdot m} = \frac{m+1}{2} \quad .$$

Lemma 4 *If we have a troubleshooting sequence* $\mathbf{s}$ *solving the troubleshooting task defined by reduction 2.1, then* $ECR(\mathbf{s}) \geq \frac{m+1}{2}$. *If there exists an action* A_b *addressing only two or fewer unsolved faults (i.e. violating the disjunction of the corresponding 3-sets) then the value of* $ECR(A_1, \ldots, A_b, \ldots, A_r)$, *is strictly greater than* $\frac{m+1}{2}$.

Proof. If a strategy $\mathbf{s} = (A_1, \ldots, A_r, CS)$ contains a call service action at the end, then at least one fault F was not addressed since all addressed faults are solved with probability 1. Therefore:

$$ECR(\mathbf{s}) \geq p(A_1 = no, \ldots, A_r = no) \cdot c_{CS} \geq p(F) \cdot c_{CS} = \frac{1}{3m} \cdot 3m(m+1) > \frac{m+1}{2} \quad .$$

Thus we may consider only sequences that are certain to solve the problem without calling the service.

The probability of taking the first step of troubleshooting is 1. Since this step solves at most 3 faults, the probability of taking the second step is at least $1 - \frac{3}{3m} = 1 - \frac{1}{m}$. Similarly, the probability of taking the i^{th} step is at least $1 - \frac{(i-1)}{m}$. Now we insert these estimates into the *ECR* calculation:

$$ECR(A_1, \dots, A_r)$$

$$= \sum_{i=1}^{r} p(A_1 = no, \dots, A_{i-1} = no) \cdot c_i$$

$$\geq \sum_{i=1}^{m} \left(1 - \frac{(i-1)}{m}\right) = m - \frac{1}{m} \cdot \sum_{i=0}^{m-1} i = \frac{m+1}{2} \quad .$$

If there exists an action A_b solving only two or fewer faults, we need at least $m+1$ steps to be certain to solve the problem. The probability of taking the step i, $i > b$ is greater than or equal to $\left(1 - \frac{2+3\cdot(i-2)}{3\cdot m}\right)$, so we conclude:

$$ECR(A_1, \dots, A_b, \dots, A_r)$$

$$\geq \sum_{i=1}^{b} \left(1 - \frac{(i-1)}{m}\right) + \sum_{i=b+1}^{m+1} \left(1 - \frac{2 + 3 \cdot (i-2)}{3 \cdot m}\right)$$

$$\geq \frac{m+1}{2} + \frac{1}{3m} + \left(\frac{1}{3} - \frac{b}{3m}\right)$$

$$\geq \frac{m+1}{2} + \frac{1}{3m} > \frac{m+1}{2}$$

since $m \geq b$.

Theorem 1 *Suppose we are given a troubleshooting problem with dependent actions, single fault assumption, and a constant $K \in \Re^+$. The decision whether there exists a troubleshooting sequence* $\mathbf{s}$ *with $ECR(\mathbf{s}) \leq K$ is a NP-complete problem*[1].

Proof. The problem is nondeterministically polynomial (NP): Let the computation starts nondeterministically on any possible sequence $\mathbf{s}$. We calculate $ECR(\mathbf{s})$ and compare whether $ECR(\mathbf{s}) \leq K$. To calculate $ECR(\mathbf{s})$ of a sequence is polynomial time. If at least one computation finds $ECR(\mathbf{s}) \leq K$ then we have a required strategy, otherwise one does not exist.

To prove the problem is NP-hard we reduce the *exact cover by 3-sets* to troubleshooting. Lemmas 3 and 4 show us that there exists a troubleshooting sequence with $ECR(\mathbf{s}) \leq \frac{m+1}{2}$ if and only if there exists an *exact cover by 3-sets*. Exact cover is a well known NP-hard problem so the troubleshooting is also NP-hard.

2.3
Independent faults

If we consider that more faults can occur simultaneously, we must define a model of their dependencies. The simplest model is to assume the faults to be independent. As compared to troubleshooting with single fault assumption, this leads to different values of expected cost of repair. On the other hand, the complexity theorems and solution methods are similar.

We aim to prove the NP-hardness of troubleshooting with dependent actions and independent faults.

Suppose the probability of any fault being present equals $p(F)$ then the probability that the fault is not present is $p(\neg F) = 1 - p(F)$.

Troubleshooting is initiated only if there is evidence of system failure. The probability of system failure is $p_0 = p(\text{system is faulty}) = 1 - p(\neg F)^{3m}$.

We denote ECR_M the expected cost of repair based on the independent faults assumption. It has different values from the ECR based on the single fault case.

The lemmas for independent faults look similar to the lemmas for a single fault.

Lemma 5 *If we have an exact cover by 3-sets $V = \{S_{j_1}, \dots, S_{j_m}\}$, then the ECR_M of the corresponding action sequence $A_{j_1}, \dots, A_{j_m}$ (in any order) is*

$$ECR_M = \frac{m}{1 - p(\neg F)^{3m}} - \frac{1}{p(\neg F)^{-3} - 1} \quad .$$

Proof. The probability of taking step $(k+1)$ is equal to one minus the probability of all not-checked faults not being present. Since no two actions solve the same fault and since in step k exactly $3k$ faults were checked, $3m - 3k$ faults remain. The probability of taking step $(k+1)$ given system failure is $\frac{1-p(\neg F)^{3m-3k}}{p_0}$.

$$ECR_M(A_{j_1}, \dots, A_{j_m})$$

$$= \sum_{i=1}^{m} p(A_{j_1} = no, \dots, A_{j_{(i-1)}} = no) \cdot c_{j_i}$$

Therefore

$$ECR_M(A_{j_1}, \dots, A_{j_m})$$

$$= \sum_{k=0}^{m-1} \frac{1 - p(\neg F)^{3m-3k}}{p_0} \cdot 1$$

$$= \frac{m}{p_0} - \frac{p(\neg F)^{3m}}{p_0} \cdot \sum_{k=0}^{m-1} p(\neg F)^{-3k}$$

$$= \frac{m}{p_0} - \frac{p(\neg F)^{3m}}{p_0} \cdot \frac{(p(\neg F)^{-3})^m - 1}{p(\neg F)^{-3} - 1}$$

$$= \frac{m}{1 - p(\neg F)^{3m}} - \frac{1}{p(\neg F)^{-3} - 1} \quad .$$

Lemma 6 *If we have a troubleshooting sequence* $\mathbf{s}$ *solving the troubleshooting task defined by reduction 1, then $ECR_M(\mathbf{s}) \geq \frac{m}{1-p(\neg F)^{3m}} - \frac{1}{p(\neg F)^{-3}-1}$. If there exists an action A_b addressing only two or fewer unsolved faults, then $ECR_M(A_1, \dots, A_b, \dots, A_r)$ is strictly greater than $\frac{m}{1-p(\neg F)^{3m}} - \frac{1}{p(\neg F)^{-3}-1}$.*

Proof. The probability of taking the first step of troubleshooting given a system fault is 1. Since this step solves at most 3 faults, the probability of taking the second step is at

---

[1] Of course, for $K < \min\{c_i\}$ or $K > \sum c_i$ the solution is trivial. The theorem says that any **general** algorithm solving these problems is NP-hard.

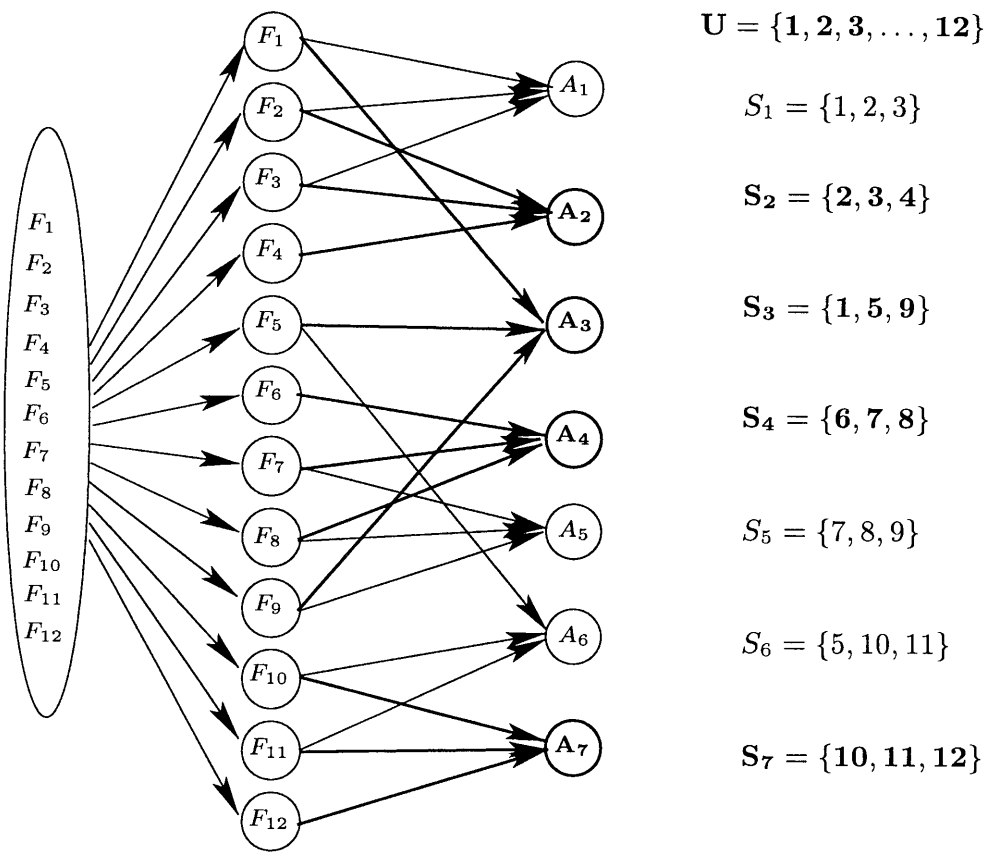

Fig. 3. Troubleshooting model of the *exact 3-sets cover*. Every action corresponds to one set, the bold-face sets are the exact cover and corresponding actions belong to the best troubleshooting sequenced

least $\frac{1}{p_0}(1 - p(\neg F)^{3m-3})$. Similarly, the probability of taking the i^{th} step is at least $\frac{1}{p_0}(1 - p(\neg F)^{3m-3(i-1)})$. Now we insert these estimates into the ECR_M calculation:

$$\begin{aligned} &ECR_M(A_1,\dots,A_r) \\ &\quad \geq \sum_{i=1}^{r} p(A_1 = no,\dots,A_{i-1} = no)\cdot c_i \\ &\quad \geq \frac{1}{p_0}\cdot\sum_{k=1}^{m}\left(1 - p(\neg F)^{3m-3(k-1)}\right) \\ &\quad \geq \frac{m}{1-p(\neg F)^{3m}} - \frac{1}{p(\neg F)^{-3}-1} \quad . \end{aligned}$$

If there is an action A_b solving only two or fewer faults, we need at least $m+1$ steps to be certain to solve the problem. The probability of taking step i, $i > b$ is greater than or equal to $\frac{1}{p_0}(1 - p(\neg F)^{3m-3(i-2)-2})$ so we conclude:

$$\begin{aligned} &ECR_M(A_1,\dots,A_b,\dots,A_r) \\ &\quad \geq \frac{1}{p_0}\cdot\left[\begin{array}{l} \sum_{k=1}^{b}\left(1 - p(\neg F)^{3m-3(k-1)}\right) + \\ \sum_{k=b+1}^{m+1}\left(1 - p(\neg F)^{3m-3(k-2)-2}\right) \end{array}\right] \\ &\quad > \frac{m}{1-p(\neg F)^{3m}} - \frac{1}{p(\neg F)^{-3}-1} \quad . \end{aligned}$$

Theorem 2 *Given a troubleshooting problem with dependent actions and independent faults assumption and a constant $K \in \Re^+$, the decision whether there exists a troubleshooting sequence* $\mathbf{s}$ *with* $ECR(\mathbf{s}) \leq K$ *is a NP-complete problem.*

Proof. Again, we check the correct sequence in polynomial time.

Lemmas 5 and 6 show us that any *exact cover by 3-sets* problem may be reduced to troubleshooting.

2.4
Polynomial problems

Troubleshooting with one fault per action is known to be polynomially solvable for both single fault assumption and independent faults [10]. There are other troubleshooting tasks that are also solvable in polynomial time.

Theorem 3 *Let us assume a troubleshooting with n faults $F_1,\dots,F_n$ and m actions $A_1,\dots,A_m$. Each action can solve* **one or two** *faults with probability* 1, *the rest with probability* 0. *We have either single fault assumption or independent faults. All actions have equal cost* 1, *cost of calling service is high, $c_{CS} > n\cdot(n+1)$, all faults have equal prior probability. There is a polynomial $O(n^5)$ algorithm to find the optimal sequence with minimal ECR (either ECR or ECR_M).*

We reduce this task to the *maximal matching* problem. The *maximal matching* can be solved in the time $O(n^5)$ (i.e. polynomial).

Maximal matching problem: A matching in a graph G is a subset of its edges such that no two edges meet the same vertex. The task is in a given graph to find a matching of maximum cardinality. The algorithm is described in [2].

For a given troubleshooting problem we construct a *maximal matching* problem as follows (see Fig. 4). We create a node for every fault, for each action that solves two faults we construct an edge between the nodes representing these faults. The *maximal matching* solution gives us a list of edges, we choose actions corresponding to these edges and for every unsolved fault we choose one (any) action solving this fault. This is a troubleshooting sequence with minimal ECR for both single fault and independent faults assumption. The proof is similar to the proofs of lemmas 3, 4, 5, and 6, therefore it is omitted.

2.5
Two faults per action troubleshooting

Let as assume each action to solve at most two actions and any distribution on faults. This case is more difficult. The straightforward reduction to the *optimal (weighted) matching* does not work.

Optimal matching problem: Let $G = (V, E)$ be a complete graph K_{2n} with non-negative weight function w on the edges. The task is to find a matching M with the maximal sum of weights of the edges in M. This problem is polynomially solvable.

2.5.1
Reduction

Let us try to define a reduction of the troubleshooting problem with one or two faults per action to optimal matching. For every fault in the troubleshooting we create a node; for every action solving two faults we create an edge between the nodes corresponding to its faults. We search for a function mapping the probabilities of faults to the weights on edges.

Lemma 7 *There does not exist any function $\Re^2 \rightarrow \Re$ that would extend reduction 2.5.1 to a reduction mapping an optimal troubleshooting sequence to the best optimal matching for all 2-troubleshooting tasks.*

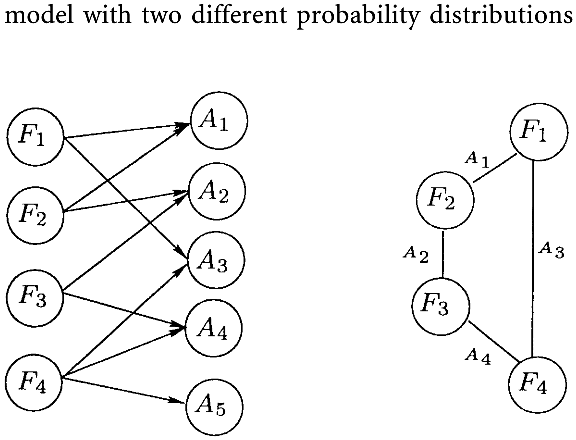

Fig. 4. Max. two faults per action troubleshooting task and corresponding *maximal matching* problem

	p'	p''
F_1	0.10	
F_2	0.15	
F_3	0.20	
F_4	0.06	
F_5	0.01	
F_6	0.01	0.05
F_7	0.05	0.01
F_8	0.42	

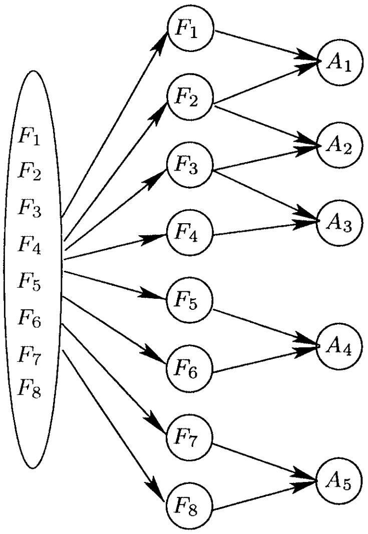

Fig. 5. Two models with the same optimal matching but different optimal sequences

Proof. The example in Fig. 5 shows a troubleshooting model with two different probability distributions of faults. The second distribution has the same values for the first four faults; but it has $p''(F_6) = 0.01$, $p''(F_7) = 0.05$ instead of $p'(F_6) = 0.05$, $p'(F_7) = 0.01$. Probabilities $p(F_1), \ldots, p(F_4)$ are equivalent in both models and matching on $F_5, \ldots, F_8$ is unique therefore the optimal matching is the same in both models.

There are two candidates for optimal troubleshooting sequences: $\mathbf{s}_{5314} = \{A_5, A_3, A_1, A_4\}$ and $\mathbf{s}_{52134} = \{A_5, A_2, A_1, A_3, A_4\}$. In the first model

$$ECR'(\mathbf{s}_{5314}) = 1.82 > ECR'(\mathbf{s}_{52134}) = 1.81 \ ,$$

while in the second model

$$ECR''(\mathbf{s}_{5314}) = 1.94 < ECR''(\mathbf{s}_{52134}) = 1.97 \ .$$

It means that each distribution leads to a different optimal troubleshooting sequence despite the fact that both models have the same optimal matching.

We proved that troubleshooting with uniform distribution on faults, one or two faults per action and the probabilities zero or one is polynomially solvable. We conjecture the general troubleshooting with one or two faults per action is NP-hard. This is similar to the relation of the 2SAT and MAX2SAT problems. The 2SAT problem is known to be polynomial whereas MAX2SAT is NP-hard [6].

3
Search for an optimal strategy

In the previous section we proved that the general troubleshooting task is NP-hard, which means that there is no polynomial algorithm providing an optimal strategy unless $P = NP$. Nevertheless, some heuristics may direct the search so that we get an optimal strategy within a reasonable time. This section summarizes our efforts in this direction.

Every troubleshooting problem can be represented by a *decision tree*. Let n and m be two different nodes in such a decision tree. If evidence corresponding to node n and m equal, i.e. $\mathbf{e}_n = \mathbf{e}_m$ then optimal strategies $\mathbf{s}^\star(\mathbf{e}_n) = \mathbf{s}^\star(\mathbf{e}_m)$.

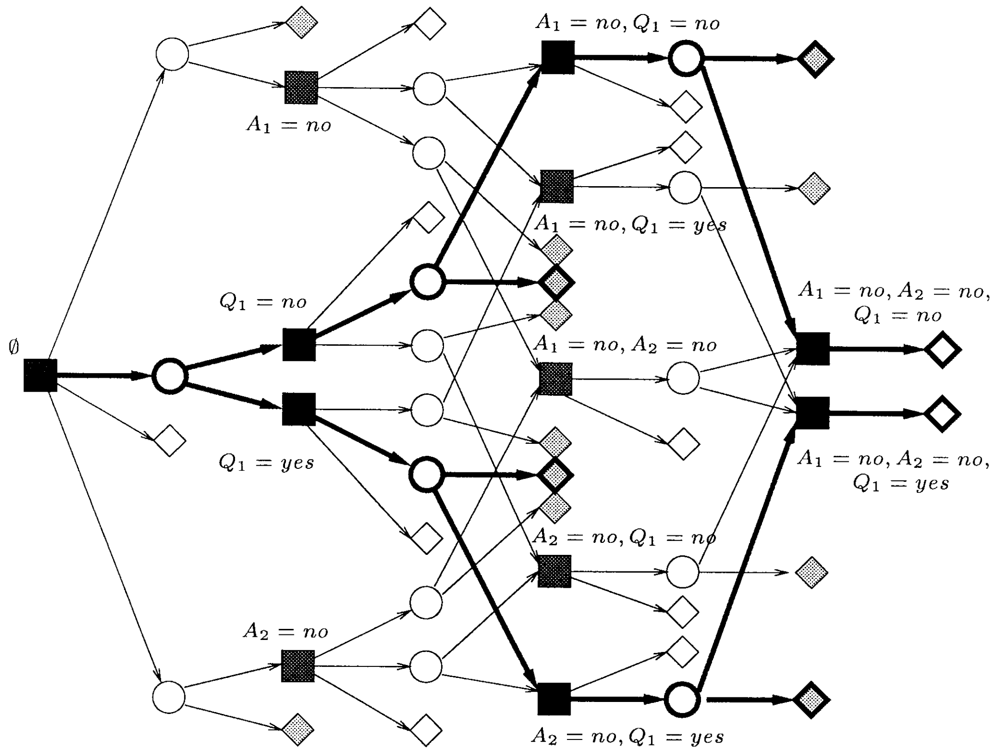

Fig. 6. Coalesced decision tree corresponding to the troubleshooting problem consisting of two actions and one question. One troubleshooting strategy is highlighted

Therefore all decision nodes with the same evidence can be coalesced to a single node, so that we get a *coalesced decision tree*. All decision trees in the following text will be coalesced. The coalesced decision tree for the Light Print Example is given in Fig. 6. For simplicity, action A_3 is omitted so that the troubleshooting problem consists of two actions A_1, A_2 and one question Q_1 only.

The decision tree of a troubleshooting problem represents all possible troubleshooting strategies. As compared with a single strategy there is one more node type – the *decision node*. Every decision node corresponds to the decision that is made when the next troubleshooting step is being chosen. Decision nodes are denoted by squares. In Fig. 6, one possible troubleshooting strategy is highlighted. There are 40 different troubleshooting strategies represented by the decision tree in Fig. 6. We have omitted the chance node labels in the figure. Decision nodes are labeled by corresponding evidence.

3.1
Correspondence to AND/OR graphs

Every decision tree of a troubleshooting problem can be interpreted as an *AND/OR graph*. An AND/OR graph of a troubleshooting problem has the same nodes and edges as the corresponding decision tree. The only difference is the interpretation assigned to the nodes. Again there are three types of nodes: *OR nodes*, *AND nodes*, and *terminal nodes* in an AND/OR graph. The chance nodes of the decision tree correspond to the AND nodes, since all their children must be included in a troubleshooting strategy. The decision nodes correspond to the OR nodes, since exactly one of their children must be included in a troubleshooting strategy. Finally, terminal nodes have equivalent interpretation in both graph types.

Propsition 1 A subgraph $\mathbf{s}$ of the AND/OR graph corresponds to a troubleshooting strategy if it fulfills the following conditions:

- $\mathbf{s}$ is a tree with the same root ϑ as the AND/OR graph
- If n is an OR node of $\mathbf{s}$, then exactly one of its children belongs to $\mathbf{s}$.
- If n is an AND node of $\mathbf{s}$, then all its children belong to $\mathbf{s}$ as well.
- Every leaf n of $\mathbf{s}$ is a terminal node.

In [7] the conditions listed above define the *solution tree* of an AND/OR graph. Consequently, if the AND/OR graph corresponds to the decision tree, then there is one-to-one correspondence between the troubleshooting strategies and the solution trees. Therefore algorithms used for solving AND/OR graphs can be used to find optimal troubleshooting strategies. In particular, the search for an optimal troubleshooting strategy is equivalent to the search for the cheapest solution tree of the AND/OR graph corresponding to a troubleshooting problem. Three algorithms will be discussed later in this section.

3.2
Depth-first search

We have implemented two versions of the *depth-first search* algorithm - with and without memory. The depth-first search algorithm without memory corresponds to the search in the not coalesced decision tree. In this case equivalent subtrees are searched through many times.

If the search is performed in the coalesced decision tree and the minimal values of $ECR^{\star}(\mathbf{e}_n)$ are stored for every expanded node n, then the complexity of the search is substantially reduced. Please note that when using the recurrent formula (4) to compute $ECR^\star(\mathbf{e}_n)$ for any node n, we need to know only the $ECR^\star$ values of the children of n in the optimal strategy $\mathbf{s}^\star_n$. The number of explored nodes for runs on three troubleshooting problems for the two versions of the depth-first search are compared in Table 5.

Another option is to reverse the search from *top-down* to *bottom-up* and systematically evaluate all nodes using the recurrent formula (4) again. Such an algorithm may be understood as an application of *dynamic programming* [5].

In spite of the substantial reduction due to the storage of the $ECR^\star$ values of the explored subtrees, the complexity of the depth-first search algorithm with memory is still very high, $O(2^{|\mathscr{A}|+|\mathscr{Q}|})$. The question arises whether we can successfully apply classical heuristic search algorithms. Next, we will discuss two algorithms: a branch & bound algorithm and the $A^\star$ algorithm generalized for AND/OR graphs (in [7] it is called $AO^\star$ algorithm). For both algorithms a heuristic function, namely an estimate of $ECR^\star$, is essential.

3.3
Heuristic function

We propose a heuristic function that exploits the conditional independence of all actions and questions given the device fault. This is the assumption already used when building the Bayesian network. Please recall that for every $F \in \mathscr{F}$ strategy $\mathbf{s}^\star(\mathbf{e} \cup F = yes)$ denotes an optimal strategy given the evidence $\mathbf{e} \cup F = yes$ and $ECR^\star(\mathbf{e} \cup F = yes)$ provides the ECR of this strategy.

Definition 4 Let $\mathscr{E}$ denote the set containing all possible evidence. The function $\underline{ECR} : \mathscr{E} \mapsto \Re^+$ is defined for each $\mathbf{e} \in \mathscr{E}$ by

$$\underline{ECR}(\mathbf{e}) = \sum_{F\in\mathscr{F}} p(F = yes \mid \mathbf{e}) \cdot ECR^\star(\mathbf{e} \cup F = yes) \ . \tag{6}$$

The following theorem claims that function $\underline{ECR}$ is an optimistic estimate of $ECR^\star$, i.e. its values are always lower than or equal to the true values of $ECR^\star$. This allows the application of the estimate into the heuristic search methods.

Theorem 4 *The function* $\underline{ECR}(\mathbf{e})$ *is an optimistic estimate of ECR of an optimal troubleshooting strategy given evidence* $\mathbf{e}$.

Proof. Let the root of $\mathbf{s}^\star(\mathbf{e})$ be n, i.e. $\mathbf{e}_n = \mathbf{e}$ and $\mathscr{L}(\mathbf{e}_n)$ be an abbreviation of $\mathscr{L}(\mathbf{s}^\star(\mathbf{e}_n))$ denoting the set of leaves of strategy $\mathbf{s}^\star(\mathbf{e}_n)$. Using Definition 2 we can write (please recall that $c(\mathbf{e}_\ell)$ denotes is a penalty applied in terminal node ℓ):

$$\begin{aligned}
ECR^\star(\mathbf{e}) &= ECR(\mathbf{s}^\star(\mathbf{e}_n) \mid \mathbf{e}_n)\\
&= \sum_{\ell\in\mathscr{L}(\mathbf{e}_n)} p(\mathbf{e}_\ell \mid \mathbf{e}_n) \cdot (c(\mathbf{e}_\ell) + t(n,l))\\
&= \sum_{\ell\in\mathscr{L}(\mathbf{e}_n)} \left(\sum_{F\in\mathscr{F}} \begin{array}{l} p(\mathbf{e}_\ell \mid \mathbf{e}_n \cup F = yes) \\ \cdot p(F = yes \mid \mathbf{e}_n) \end{array} \right) \cdot (c(\mathbf{e}_\ell) + t(n,l))\\
&= \sum_{F\in\mathscr{F}} p(F = yes \mid \mathbf{e}_n) \cdot \sum_{\ell\in\mathscr{L}(\mathbf{e}_n)} \begin{array}{l} p(\mathbf{e}_\ell \mid \mathbf{e}_n \cup F = yes) \\ \cdot (c(\mathbf{e}_\ell) + t(n,l)) \end{array}\\
&= \sum_{F\in\mathscr{F}} p(F = yes \mid \mathbf{e}_n) \cdot ECR(\mathbf{s}^\star(\mathbf{e}_n) \mid \mathbf{e}_n \cup F = yes) \ .
\end{aligned}$$

In the previous formula $ECR(\mathbf{s}^\star(\mathbf{e}_n) \mid \mathbf{e}_n \cup F = yes)$ provides the ECR of strategy $\mathbf{s}^\star(\mathbf{e}_n)$ which need not be optimal when having the given evidence $\mathbf{e}_n \cup F = yes$. Let $\mathbf{s}^\star(\mathbf{e}_n \cup F = yes)$ be the optimal strategy given evidence $\mathbf{e}_n \cup F = yes$. It is obvious that for $F \in \mathscr{F}$

$$\begin{aligned}
&ECR(\mathbf{s}^\star(\mathbf{e}_n) \mid \mathbf{e}_n \cup F = yes)\\
&\geq ECR(\mathbf{s}^\star(\mathbf{e}_n \cup F = yes) \mid \mathbf{e}_n \cup F = yes)
\end{aligned}$$

and consequently the following inequality holds:

$$\begin{aligned}
&ECR^\star(\mathbf{e})\\
&\quad \geq \sum_{F\in\mathscr{F}} p(F = yes \mid \mathbf{e}) \cdot ECR^\star(\mathbf{e} \cup F = yes)\\
&\quad \geq \underline{ECR}(\mathbf{e}) \ ,
\end{aligned}$$

which corresponds to the assertion of the theorem.

Table 5. Number of expanded nodes for the depth-first search algorithms with and without memory

Depth-first search	$\lvert\mathscr{A}\rvert = 6$, $\lvert\mathscr{Q}\rvert = 2$	$\lvert\mathscr{A}\rvert = 9$, $\lvert\mathscr{Q}\rvert = 3$	$\lvert\mathscr{A}\rvert = 12$, $\lvert\mathscr{Q}\rvert = 3$
Without memory	5,060	130,328	476,191
With memory	374	4,354	16,881

A basic advantage of this estimate is that computation of $ECR^\star(\mathbf{e} \cup F = yes)$ does not require expensive operations if the actions are conditionally independent given the device fault. If the fault is known then there are usually only few actions that may fix the problem, i.e. $p(A = yes \mid F = yes) > 0$. For every fault $F \in \mathscr{F}$ the actions that are not contained in the evidence $\mathbf{e}$ are ordered according to $p(A = yes \mid F = yes)/c_A$ which gives an optimal strategy for the given evidence $\mathbf{e} \cup F = yes$. Please note that the value of the ECR estimate is computed using the conditional probabilities $p(A = yes \mid F = yes)$ that are already available from the original model. In the light of the new evidence $\mathbf{e}$ it is only necessary to update the probabilities of the faults $p(F)$.

We have performed experiments on nine different models designed by domain experts for troubleshooting laser printers. We measured how far the lower bound was from the optimal value of ECR. We computed

$$\varrho_k = 100 \cdot \frac{ECR^\star(\mathbf{e}_k) - \underline{ECR}(\mathbf{e}_k)}{ECR^\star(\mathbf{e}_k)} \ .$$

For every tested model we computed the average value $\bar{\varrho}$ over all nodes in the decision tree. In most cases the estimate was quite close to the true values of $ECR^\star$, the average relative difference was $\bar{\varrho} = 10\%$. For one tested model the value of $\bar{\varrho}$ was 2.5%, and for two models it was far from the optimal value, $\bar{\varrho} = 45\%$ and $\bar{\varrho} = 43\%$.

3.4

Branch & bound algorithm

Our implementation of branch & bound algorithm performs depth first search with pruning. The temporarily best $ECR'(\mathbf{e}_n)$ is stored for every expanded decision node n. Pruning of an edge coming from n is performed as soon as it is certain that any strategy that include the edge cannot be the optimal one. Next, we describe the pruning formally.

Let n be a decision node, and m one of its children in the decision tree. Let $children(m) = \{m_1, \ldots, m_r\}$ be the set of children of node m corresponding to evidence $\mathbf{e}_{m_i} = \mathbf{e}_m \cup S = s_i$ where S is a troubleshooting step with outcomes $\{s_1, \ldots, s_r\}$. Further assume that the value of $ECR^\star(\mathbf{e}_m)$ has already been computed for the children $m_1, \ldots, m_q$ and for the remaining children $m_{q+1}, \ldots, m_r$ the value of $\underline{ECR}(\mathbf{e}_m)$ is to be provided. The edge from n to m is pruned immediately after it is realized that

$$\begin{aligned} ECR'(\mathbf{e}_n) \leq c_S &+ \sum_{i=1}^{q} P(S = s_i \mid \mathbf{e}_n) \cdot ECR^\star(\mathbf{e}_{m_i}) \\ &+ \sum_{i=q+1}^{r} P(S = s_i \mid \mathbf{e}_n) \cdot \underline{ECR}(\mathbf{e}_{m_i}) \ . \end{aligned}$$

Since the applied function $\underline{ECR}$ is an optimistic estimate of $ECR^\star$, the optimum solution must be reached.

3.5

AO* algorithm

AO* algorithm is a version of the well known A* algorithm designed to solve AND/OR graphs (see [7] for details). Our version of the algorithm traverses the coalesced decision tree, referred to as *coalesced_dt* in the algorithm. In Fig. 6 an example of a coalesced decision tree is provided. Let ϑ denote the root of the coalesced decision tree. A troubleshooting strategy, referred to as *strategy* is continuously being constructed. Fig. 2 provides an example of a troubleshooting strategy. The main part of AO* algorithm is described in Table 6.

Table 6. AO* algorithm

```
$queue := \emptyset$;
$frontier := \{\vartheta\}$;
$complete := false$;
repeat
  /* a frontier node selection */
  $n := select\_node(frontier)$;
  $queue.push(n)$;
  /* expansion of node $n$, backward propagation */
  while $queue \neq \emptyset$ do
        $update\_decision(queue.pop\_node)$;
  /* tracing up a strategy, frontier creation */
  $strategy.nodes := \{best\_child[\vartheta]\}$;
  $strategy.edges := \emptyset$;
  $frontier := \emptyset$;
  $trace\_up\_strategy(best\_child[\vartheta])$;
 until $frontier = \emptyset$;
 $return(strategy)$;
```

In the first part of the main cycle of the AO* algorithm a node from the list *frontier* is chosen. The list *frontier* contains untouched decision nodes that are children of leaves of the current best strategy. The existence of such a decision node implies that the strategy is not complete yet.

Function *select_node* arbitrarily selects one of the frontier nodes. However, if additionally to the lower bound $\underline{ECR}(\mathbf{e})$ an upper bound of $\overline{ECR}(\mathbf{e})$ was computed for each decision node then probably a better option than a random selection would be the selection of a frontier node having the largest difference of the lower and upper bounds. Its expansion may bring the most information to restrict the interval for the true value of $ECR^\star(\mathbf{e})$.

In Table 7 procedure $update\_decision(n)$ is presented. For each untouched grand child of the selected decision node n (i.e. child of its child) the lower bound estimate $\underline{ECR}(\mathbf{e}_k)$ is computed. Based on the lower bounds a lower bound estimate is computed for each child ℓ of node n. The child with the lowest value of $ECR[\ell]$ is the best child of n. It is denoted as $\ell^\star$ and stored in $best\_child[n]$. The value of $ECR[\ell^\star]$ is stored in $ECR[n]$. Symbols $pa(n)$ and $ch(n)$ stand for the set of parents of node n and the set of children of node n, respectively.

Not only node k but also all preceding decision nodes in the *coalesced_dt* that can be influenced by the new value of $ECR[k]$ have to be updated. Nodes that need to have their lower bounds updated are added to the *queue*. A queue means that the first added node is selected first. The *queue* guarantees that a node is updated only after all its touched descendants in the coalesced decision tree.

When all required nodes are updated then the current best strategy can be traced up. See Table 8 where the procedure *trace_up_strategy* is described. The array *best_child* containing best child for each touched decision

Table 7. Procedure $update\_decision(n)$

```
$touch(n)$;
for each $\ell \in ch(n)$ do
  if $is\_terminal\_node(\ell)$ then $ECR[\ell] := c(\mathbf{e}_\ell)$
  else    /* $\ell$ is a chance node */
     if $untouched(\ell)$ then
          $touch(\ell)$;
          $ECR[\ell] := c_{step(\ell)}$;
          for each $k \in ch(l)$ do
              if $is\_decision\_node(k)$ then
                    if $untouched(k)$ then
                        $touch(k)$;
                        $ECR[k] := \underline{ECR}(\mathbf{e}_k)$;
                        for each $p \in pa(k) \setminus \ell$ do
                            if $touched(p)$ and $pa(p) \notin queue$ then
                                 $queue.push(pa(p))$;
                $ECR[\ell] := ECR[\ell] + p(\mathbf{e}_k \mid \mathbf{e}_\ell)^* ECR[k]$;
  $\ell^* := \arg\min_{\ell \in ch(n)} ECR[\ell]$;
  $ECR[n] := ECR[\ell^*]$;
  $best\_child[n] := \ell^*$;
  for each $p \in pa(n)$ do
        if $touched(p)$ and $pa(p) \notin queue$ then
            $queue.push(pa(p))$;
  $return()$;
```

Table 8. Procedure *trace_up_strategy*(n)

```
for each $m \in ch(n)$ do
    if $is\_terminal\_node(m)$ then
        $strategy.nodes := strategy.nodes \cup \{m\}$;
        $strategy.edges := strategy.edges \cup \{n \rightarrow m\}$;
    else    /* $m$ is a decision node */
        if $touched(m)$ then
            $strategy.nodes := strategy.nodes$
                               $\cup\ best\_child[m]$;
            $strategy.edges := strategy.edges$
                               $\cup\ (n \rightarrow best\_child[m])$;
            $trace\_up\_strategy(best\_child[m])$;
        else
            $frontier := frontier \cup \{m\}$;
$return()$;
```

node is used to construct quickly the current best strategy. It is described by a list of edges and a list of nodes. At the same time the list of frontier nodes of the strategy is created. It is a consequence of a well-known property of $A^\star$ algorithms that if an optimistic estimate of ECR is used (in our case it is $\underline{ECR}(\mathbf{e})$) then the first expanded complete strategy is an optimal strategy.

Remark 3 The AO* algorithm can be used as an *anytime algorithm*. The condition in the main cycle of the AO* algorithm is extended so that it also checks whether there is a request for a best next troubleshooting step. If this is the case, then the algorithm returns the root of *strategy*, say node r, and continues the search disregarding all children of root ϑ of *coalesced_dt* except the child r corresponding to the recommended troubleshooting step $step(r)$. When the outcome of step $step(r)$ is known then the root ϑ of *coalesced_dt* is redefined to be the child of node r corresponding to the observed outcome of $step(r)$ and search for a best strategy continues until new request for the next troubleshooting step appears.

3.6
Approximative methods

Whatever the efficiency of the proposed heuristic algorithms searching for the optimal strategy they only enable us to determine optimal strategies for domains with less than 20 troubleshooting steps. Therefore methods that provide reasonably good troubleshooting strategies in real-time seem to be necessary.

The Bayesian Automated Troubleshooting System (BATS) was developed in the Laboratory for Normative Systems, a joint Hewlett-Packard Company and Aalborg University project. The basic idea of this method is that a local computation is performed whenever a new troubleshooting step has to be chosen. It is assumed that the single fault assumption holds. The BATS approach, which we are not going to discuss in detail here, exploits several heuristics based on the p/c ratio. For further information, see [9].

In [4] the strategies obtained by use of suboptimal methods were compared with the optimal solutions. The comparisons were performed for nine models designed by domain experts for troubleshooting of laser printers. The algorithm used to find the optimal solutions was depth-first search with memory, discussed in this paper in Section 3. The algorithm was implemented in C^{++}, partly by a DAT3 student group within a student project at Aalborg University [8] and partly by the authors. The ECR values of the strategies provided by the BATS troubleshooter were very close to the optimal values. Thus, in the case of the troubleshooting of laser printers it turns out that good troubleshooting strategies can be provided in real time despite the fact that the search for an optimal strategy is NP-hard.

4
Conclusions

In this paper we have proved that the general troubleshooting task is NP-hard. Troubleshooting with questions and independent actions is NP-hard as well [12]. Even though the precise border between polynomial and NP-hard troubleshooting problems is not known, we were able to narrow the area of uncertainty. In [11] the complexity of different troubleshooting problems is studied in detail.

The proposed heuristic methods used to find an optimal strategy appeared to be applicable only to domains with a limited number of actions and questions. A satisfactory solution can be to use an approximative method either working on-line or off-line. Off-line approximative methods find a full suboptimal strategy before a user actually performs the troubleshooting. Examples of on-line algorithms are the *anytime* version of the AO* algorithm or the BATS troubleshooter [9]. In the case of the troubleshooting of laser printers the suboptimal strategies provided by the BATS troubleshooter are not far from optimal ones [4].

References

1. **Bellman R, Dreyfus S** (1962) Applied Dynamic Programming. Princeton University Press, Princeton, N.J.
2. **Jack Edmonds** (1965) Paths, trees and flowers. Canardian Journal of Mathematics, **17**: 449–467
3. **Heckerman D, Breese JS, Rommelse K** (1995) Decision theoritic troubleshooting. Commuinications of the ACM, **38**(3): 49–57
4. **Finn V Jensen, Uffe Kjaerulff, Brian Kristiansen, Helge Langseth, Claus Skaanning, Jiří Vomlel, Marta Vomlelová** (2001) The SACSO methodology for troubleshooting complex systems. Artificial Intelligence for Engineering Design, Analysis and Manufacturing Vol. 15, pp. 321–333
5. **Kalagnanam J, Henrion M** (1988) A Comparison of decision analyais and expert rules for sequential analysis. In: Besnard P, Hanks S (eds) The Fourth Conference on Uncertainty in Artificial Intelligence, pp. 271–281, New York
6. **Christos H. Papadimitriou** (1994) Computational complexity. Addision-Wesley Publishing Company
7. **Pear J** (1984) Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addision-Wesley Publishing Company, Inc.
8. **Rasmussen EB, Pedersen MH, Christiansen MK, Pedersen ST** (1999) How to pull a solution tree out of a Bayesian Network (DAT3 report). Aalborg University, Department of Computer Science
9. **Skaaning C, Jensen FV, Kjaerulff U** (2000) Printer troubleshouting using Bayesian networks. In the Thirteenth International Conference on Industrial and Engineering Applications of AI & Expert Systems

10. **Srinivas S** (1995) A polynomial algorithm for computing the optimal repair strategy in a system with independent component failures. In: Proc. of the Eleventh Conference on Uncertainty in Artificial Intelligence, pp. 515–522, Montréal, Canada
11. **Marta Vomlelová** (2001) Decision Theoretic Troubleshooting. PhD thesis, University of Economics, Prague, Czech Republic
12. **Marta Vomlelová** (2001) Complexity of decision-theoretic troubleshooting. In: Proceeding of 9th Spanish Association for Artificial Intelligence Conference and 4th Technological Transfer of Intelligence Artificial Congress, Gijón, Spain, November 14–16 Accepted