The SACSO methodology for troubleshooting complex systems

Finn V. Jensen[1] Uffe Kjærulff[1] Brian Kristiansen[2]
Helge Langseth[1] Claus Skaanning[2] Jiří Vomlel[1]
Marta Vomlelová[1]

[1]Department of Computer Science
Aalborg University, Denmark

[2]Hewlett-Packard Laboratory for Normative Systems, Denmark

22 December 2000

Abstract

The paper describes the task of performing efficient decision-theoretic troubleshooting of electro-mechanical devices. In general, this task is NP-complete, but under fairly strict assumptions, a greedy approach will yield an optimal sequence of actions, as discussed in the paper. This set of assumptions is weaker than the set proposed by Heckerman et al. (1995). However, the printing system domain, which motivated the research and which is described in detail in the paper, does not meet the requirements for the greedy approach, and a heuristic method is used. The method takes value of identification of the fault into account and it also performs a partial two-step look-ahead analysis. We compare the results of the heuristic method with optimal sequences of actions, and finds only minor differences between the two.

Keywords: Troubleshooting, decision theory, Bayesian network

1 Introduction

SACSO (Systems for Automated Customer Support Operations) is a collaboration between the Research Unit of Decision Support Systems at Aalborg University and Customer Support R&D, Hewlett-Packard Company. A result of SACSO is a decision-theoretic system for troubleshooting printing systems. A printing system consists of several components: the application from which the printing command is sent, the printer driver, the network connection, the server controlling the printer, the printer itself, etc. It is a complex task to troubleshoot such a system, and the printer industry spends millions of dollars a year on customer support. Therefore, automating the troubleshooting process is highly beneficial for customer as well as supplier.

Traditionally, computer-aided diagnoses or troubleshooting consists in using evidence to narrow down the set of possible causes for observed symptoms and to order them with respect to likelihood (de Kleer & Williams 1987). In decision-theoretic troubleshooting costs and likelihoods are balanced in order to find the next action.

Decision-theoretic troubleshooting was studied by Kalagnanam & Henrion (1990), and it was extended to the context of Bayesian networks by Heckerman et al. (1995). They provide a framework for suggesting sequences of questions, repair actions, and configuration changes to obtain further information. By calculating a local efficiency of the possible repair actions and continuously choosing the one of highest efficiency, a repair sequence is established. Assuming only a single fault, perfect repair actions, independent actions, and independent costs, the method finds the optimal sequence of actions. With respect to questions, Heckerman et al. (1995) suggest a myopic one-step lookahead.

Troubleshooting is addressed in a similar way by Srinivas (1995). In particular, he addresses the problem of multiple faults, and under the assumption of independent faults, he provides an effective way of determining an optimal repair sequence.

When troubleshooting printing systems, it is more natural to assume single fault than to assume independent faults. We exploit the single fault assumption heavily in knowledge acquisition as well as in inference: naïve Bayes models suffice, and probability updating is very fast, allowing for methods requiring a large set of updates.

However, the repair actions for printing systems are imperfect, dependent, and a myopic analysis of questions is insufficient for uncovering the value of asking a question later in the session. Therefore, we have modified the approach of Heckerman et al. (1995), taking advantage of the opportunity to perform many probability updates. The SACSO algorithms for selection of troubleshooting steps have been further described by Skaanning et al. (2000).

To allow domain experts to efficiently implement their models in practice, the SACSO project has also resulted in a knowledge acquisition tool described by Skaanning (2000). The tool, called BATS Author, allows domain expert with no knowledge of Bayesian networks to construct troubleshooting models, and thereby eliminates the traditional knowledge acquisition bottleneck for Bayesian networks.

2 The decision-theoretic troubleshooting task

A fault causing a (man-made) device to malfunction is identified and eliminated through a sequence of troubleshooting steps. Some steps are *repair steps* which may or may not fix the problem, some steps are *observation steps* which cannot fix the problem, but may give indications of the causes of the problem, and some steps have repair aspects as well as observation aspects. All steps have a cost in terms of money, time, etc. or combinations thereof. The task is to find the cheapest strategy for sequencing the troubleshooting steps. In this paper we deal with pure repair steps and pure observation steps only, and we shall call them *actions* and *questions*, respectively.

A troubleshooting problem can be represented and solved through a decision tree. However, as decision trees have a risk of becoming intractably large, we look for ways of pruning the decision tree. Also, a troubleshooting strategy may by itself be intractably large, and we look for ways of stepwise expanding the strategy through local calculations based on the actual past.

2.1 Action sequences

In this section we consider a set of steps consisting of actions only. An action, A_i, has two possible outcomes, namely "$A_i = \text{yes}$" (the problem was fixed) and "$A_i = \text{no}$" (the action failed to fix the problem). Each action, A_i, has a cost $C_{A_i}(\varepsilon)$ which may depend on evidence ε. We shall sometimes use $C_i(\varepsilon)$ (or C_i) as shorthand for $C_{A_i}(\varepsilon)$. As there are no questions, a *troubleshooting strategy* is a sequence of actions $s = \langle A_1, \ldots, A_n \rangle$ prescribing the process of repeatedly performing the next action until an action fixes the problem or the last action has been performed.

When solving a troubleshooting problem we have some initial evidence ε and in the course of executing actions in the troubleshooting sequence $s = \langle A_1, \ldots, A_n \rangle$ we collect further evidence, namely that the previous actions have failed. We let ε^i denote the evidence that the first i actions have failed, and we shall refer to a set of failed actions as *simple evidence*. In the following we shall not mention the initial evidence explicitly.

Definition 1 The *expected cost of repair*, ECR, of a troubleshooting sequence $s = \langle A_1, \ldots, A_n \rangle$ with costs C_i is the mean of the costs until an action succeeds or all actions have been performed:

$$\mathrm{ECR}(s) \equiv \sum_i \mathrm{ECR}_i(s),$$

where

$$\mathrm{ECR}_i(s) = C_i(\varepsilon^{i-1}) P(\varepsilon^{i-1}). \qquad \square$$

Note that the term "expected cost of repair" may be misleading as we allow a situation where all actions have been performed without having fixed the problem. If this happens, it will happen with the same probability no matter the sequence, and therefore we need not estimate a cost for it. We may also extend the set of actions with a *call service* action, CS. We shall return to this in Section 2.3.

Now, consider two neighboring actions A_i and A_{i+1} in s, and let s' be obtained from s by swapping the two actions. The contribution to $\mathrm{ECR}(s)$ from the two actions is

$$C_i(\varepsilon^{i-1}) P(\varepsilon^{i-1}) + C_{i+1}(\varepsilon^i) P(A_i = \text{no}, \varepsilon^{i-1}), \tag{1}$$

and the contribution to $\mathrm{ECR}(s')$ from the two actions is

$$C_{i+1}(\varepsilon^{i-1}) P(\varepsilon^{i-1}) + C_i(\varepsilon^{i-1}, A_{i+1} = \text{no}) P(A_{i+1} = \text{no}, \varepsilon^{i-1}). \tag{2}$$

As the difference between (2) and (1) equals $\mathrm{ECR}(s') - \mathrm{ECR}(s)$, we get

$$\begin{aligned} &\mathrm{ECR}(s') - \mathrm{ECR}(s) = \\ &\quad P(\varepsilon^{i-1}) \cdot \big(C_{i+1}(\varepsilon^{i-1}) - C_i(\varepsilon^{i-1}) + C_i(\varepsilon^{i-1}, A_{i+1} = \text{no}) P(A_{i+1} = \text{no} \,|\, \varepsilon^{i-1}) - \\ &\quad C_{i+1}(\varepsilon^i) P(A_i = \text{no} \,|\, \varepsilon^{i-1}) \big) \,. \end{aligned}$$

If s is an optimal troubleshooting sequence, we must have $\mathrm{ECR}(s) \leq \mathrm{ECR}(s')$, and therefore

$$\begin{aligned} & C_i(\varepsilon^{i-1}) + C_{i+1}(\varepsilon^i)P(A_i = \text{no} \mid \varepsilon^{i-1}) \leq \\ & \quad C_{i+1}(\varepsilon^{i-1}) + C_i(\varepsilon^{i-1}, A_{i+1} = \text{no})P(A_{i+1} = \text{no} \mid \varepsilon^{i-1}). \end{aligned} \tag{3}$$

If it holds that the costs are independent of the actions taken, (3) can be rewritten as

$$\frac{P(A_i = \text{yes} \mid \varepsilon^{i-1})}{C_i} \geq \frac{P(A_{i+1} = \text{yes} \mid \varepsilon^{i-1})}{C_{i+1}}. \tag{4}$$

Definition 2 Let A be a repair action and let ε be the evidence compiled so far. The *efficiency* of A is defined as

$$\mathrm{ef}(A \mid \varepsilon) = \frac{P(A = \text{yes} \mid \varepsilon)}{C_A(\varepsilon)}. \qquad \square$$

Proposition 1 *Let s be an optimal sequence of actions for which the costs are independent of the actions taken. Then it must hold that $ef(A_i \mid \varepsilon^{i-1}) \geq ef(A_{i+1} \mid \varepsilon^{i-1})$.*

In general, (3) can be used for pruning the decision tree, but Proposition 1 makes it even simpler. Assume that action B_i has been chosen at a branch where the options were $B_1, \ldots, B_m$ with current efficiencies $\mathrm{ef}(B_1 \mid \varepsilon), \ldots, \mathrm{ef}(B_m \mid \varepsilon)$. Now, if B_i fails, only B_j's for which $\mathrm{ef}(B_i \mid \varepsilon) \geq \mathrm{ef}(B_j \mid \varepsilon)$ may be chosen, but after failure of B_j any action may be chosen.

2.2 The greedy approach

It would be much easier to solve the troubleshooting problem if we could base the sequencing on a greedy approach: choose always an action with highest efficiency. However, Proposition 1 does not guarantee that this approach will yield an optimal troubleshooting sequence.

In Figure 1 there are 4 possible causes, C_1, C_2, C_3, and C_4, for a device malfunctioning, and we assume that exactly one of the causes is present, and that the prior probabilities are 0.2, 0.25, 0.40, and 0.15, respectively. Assume that all actions have cost 1. Then action A_2 has the highest efficiency, and if A_2 fails, then A_1 has higher efficiency than A_3. The sequence $\langle A_2, A_1, A_3 \rangle$ has $\mathrm{ECR} = 1.50$. However, the sequence $\langle A_3, A_1 \rangle$ has $\mathrm{ECR} = 1.45$.

To analyze why the decreasing efficiency approach does not guarantee an optimal sequence, let $\langle A_1, \ldots, A_n \rangle$ be a sequence ordered by decreasing efficiency. If the sequence is not optimal, there must be two actions A_i and A_j, $i < j$, which, in the optimal sequence, are taken in different order. At the time where A_i is chosen, we have

$$\frac{P(A_i = \text{yes} \mid \varepsilon)}{C_i} > \frac{P(A_j = \text{yes} \mid \varepsilon)}{C_j}.$$

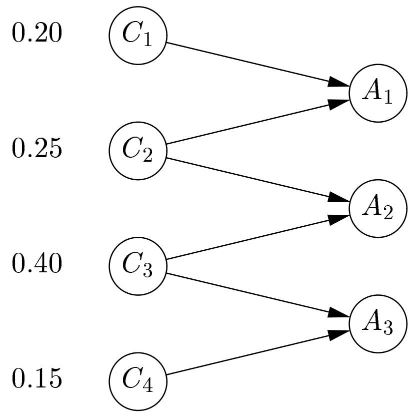

Figure 1: An example of dependent actions. Each of $C_1, \dots, C_4$ is a possible cause of a particular fault of a device, and each of the actions, $A_1, \dots, A_3$, will eliminate the fault associated with their parent causes.

In the optimal sequence, where A_j is chosen before A_i, we have

$$\frac{P(A_i = \text{yes} \,|\, \varepsilon')}{C_i} < \frac{P(A_j = \text{yes} \,|\, \varepsilon')}{C_j},$$

where ε and ε' are simple evidence (not involving A_i and A_j). We can infer that an action sequence $\langle A_1, \dots, A_n \rangle$ is optimal if for all $i < j$ it holds that

$$\text{ef}(A_j \,|\, \varepsilon) \leq \text{ef}(A_i \,|\, \varepsilon),$$

where ε is simple evidence (not involving A_i and A_j).

Proposition 2 *Consider the following assumptions.*

- *The device has n different faults $F_1, \dots, F_n$ and n different repair actions $A_1, \dots, A_n$.*
- *Exactly one of the faults is present.*
- *Each action has a specific probability of repair, $p_i = P(A_i = \mathit{yes} \,|\, F_i)$, and $P(A_i = \mathit{yes} \,|\, F_j) = 0$ for $i \neq j$.*
- *The cost C_i of a repair action does not depend on the performance of previous actions.*

If these assumptions hold, then $\mathit{ef}(A_j) \leq \mathit{ef}(A_i)$ implies that $\mathit{ef}(A_j \,|\, \varepsilon) \leq \mathit{ef}(A_i \,|\, \varepsilon)$, where ε is simple evidence (not including A_i and A_j).

Note that we do not assume the repair actions to be perfect. They may fail to fix a fault which they are supposed to fix.

Proof: Let A_m be an action which has failed. We shall calculate $P(A_i = \text{yes} \,|\, A_m = \text{no})$ (for notational convenience, we omit mentioning of the current evidence). Due to the single-fault assumption, we have $P(A_m = \text{no} \,|\, A_i = \text{yes}) = 1$. Using Bayes' rule we get

$$P(A_i = \text{yes} \,|\, A_m = \text{no}) = \frac{P(A_m = \text{no} \,|\, A_i = \text{yes}) P(A_i = \text{yes})}{P(A_m = \text{no})} = \frac{P(A_i = \text{yes})}{P(A_m = \text{no})}.$$

That is, $P(A_m = \text{no})$ is a normalizing constant for the remaining actions, and the relative order of efficiencies is preserved. □

The following theorem concludes the considerations. The theorem is a slight extension of similar results by Kalagnanam & Henrion (1990) and Heckerman et al. (1995).

Theorem 1 *Let $s = \langle A_1, \dots, A_n \rangle$ be an action sequence for a troubleshooting problem fulfilling the conditions in Proposition 2. Assume that s is ordered according to decreasing initial efficiencies. Then s is an optimal action sequence and*

$$ECR(s) = \sum_{i=1}^{n} C_i \left(1 - \sum_{j=1}^{i-1} p_j \right). \tag{5}$$

Proof: From the proof of Proposition 2, we have that the relative order of the efficiencies of the actions are preserved. For any action sequence s' which is not ordered according to $\text{ef}(A_i)$ there will be a j so that $\text{ef}(A_j) < \text{ef}(A_{j+1})$ and therefore $\text{ef}(A_j \,|\, \varepsilon^j) < \text{ef}(A_{j+1} \,|\, \varepsilon^j)$. Hence s' can be improved by swapping A_j and A_{j+1}. From the definition we have

$$\text{ECR}(s) = \sum_{i=1}^{n} C_i P(\varepsilon^i).$$

Due to the single fault assumption we have $P(\varepsilon^i) = 1 - \sum_{j=1}^{i-1} p_j$. □

2.3 Call service

The action *call service* (CS) will always solve the problem. The cost of CS is not the unknown price of fixing the device, but the possible overhead of having outsiders fixing a problem you could have fixed yourself. The efficiency of CS is $1/C_{CS}$ no matter the set of actions performed so far.

Let $s = \langle A_1, \dots, A_n \rangle$ be an optimal action sequence resulting from a situation meeting the assumptions in Proposition 2. It may be so that the sequence should be broken before A_n and service is called. According to Proposition 1, CS shall only be performed after an action of higher efficiency. In SACSO we suggest the CS action as soon as it has maximal efficiency. However, this is not guaranteed to be optimal. The question of finding an optimal action sequence including CS is of higher combinatorial complexity. Instead of looking for a sequencing of actions each of which must eventually be performed if the other actions fail, we shall now look for a subset of actions and a sequencing of them. We shall not go further into this problem.

2.4 Questions

The outcome of a question may shed light on any of the possible faults, or it may be focused on a particular fault.

The troubleshooting task is to interleave actions and questions such that the expected cost is minimal. To do so, we need to analyze the value of answers to questions.

Imagine that we are in the middle of a troubleshooting sequence; we have so far gained the evidence ε, and now we have the option to ask the question Q with cost C_Q. For simplicity, we assume that Q has only two outcomes, "yes" and "no". Assume that no matter the outcome of Q, we are able to calculate the minimal expected cost of repair for the remaining sequence. So let ECR be the minimal expected cost if Q is not performed, and let $\mathrm{ECR}_{Q=\mathrm{yes}}$ and $\mathrm{ECR}_{Q=\mathrm{no}}$ denote the same for the outcomes "yes" and "no", respectively.

Then the value of observing Q is

$$V(Q) = \mathrm{ECR} - \left(P(Q = \mathrm{yes} \,|\, \varepsilon)\mathrm{ECR}_{Q=\mathrm{yes}} + P(Q = \mathrm{no} \,|\, \varepsilon)\mathrm{ECR}_{Q=\mathrm{no}}\right), \tag{6}$$

and Q is performed if and only if $V(Q) > C_Q$.

In order to determine whether or not to ask a question prior to an action, we have to analyze all possible succeeding sequences, and if there are several actions and questions, it is in general intractable: in the future we will also have question options to interleave.

A workable approximation is the *myopic strategy*: assume at any stage of troubleshooting that we allow questions to be asked, but in the future we allow only repair actions. In that case, the task reduces to calculating expected costs given the various outcomes of the possible questions, and the approaches from the previous section can be used.

2.5 Strategy trees

When questions are part of the troubleshooting, then a troubleshooting strategy is a tree rather than a sequence. To emphasize this fact, we shall sometimes refer to such a strategy as a *strategy tree*. Figure 2 provides an example of a strategy tree.

There are two types of nodes in a strategy tree — *chance nodes* and *terminal nodes*. Chance nodes are displayed as circles, and they are labeled with troubleshooting steps (actions or questions). Edges are labeled with outcomes of the steps, and we let $\mathcal{L}(\varepsilon)$ denote the function yielding labels to edges, ε. Terminals are diamond shaped, and they indicate that the device has been repaired. The set of terminal nodes of a strategy tree s is denoted $\mathcal{L}(s)$.

Let $\mathrm{path}(n)$ be the sequence of edges constituting a path from the root node to node n in a strategy tree. Then $\varepsilon_n = \bigcup_{\varepsilon \in \mathrm{path}(n)} \mathcal{L}(\varepsilon)$ defines the evidence corresponding to the already performed actions and questions. Furthermore, let $P(\varepsilon_n)$ denote the probability of evidence ε_n, i.e., the probability of getting to node n from the root node. Finally, let $t(n)$ denote the total cost of actions and questions in the path from the root node to node n. For example, in Figure 2, the evidence corresponding to node d labeled by A_2

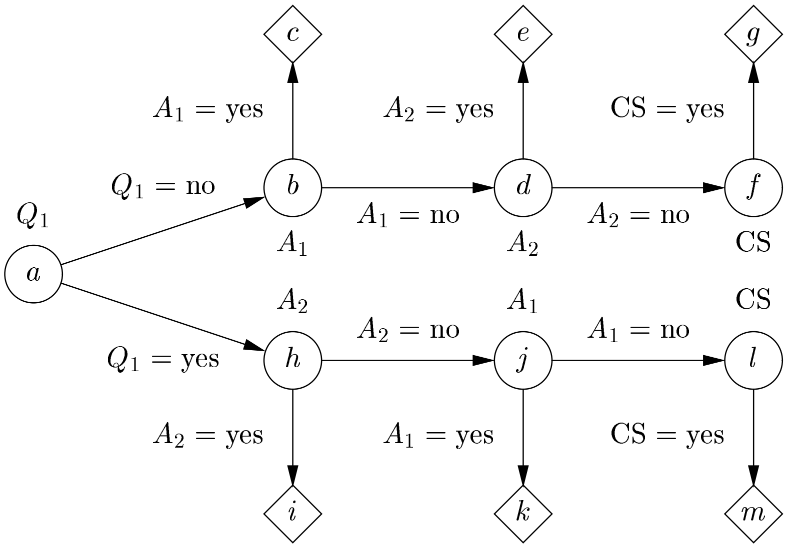

Figure 2: A strategy tree.

is $Q_1 = \text{no}$, $A_1 = \text{no}$, the probability of getting there is $P(Q_1 = \text{no}, A_1 = \text{no})$, and the total cost of getting there is $C_{Q_1} + C_{A_1}$.

Next, we extend the definition of expected cost of repair to strategy trees.

Definition 3 The expected cost of repair of troubleshooting strategy s is defined as

$$\text{ECR}(s) = \sum_{\ell \in \mathcal{L}(s)} P(\varepsilon_\ell) \cdot t(\ell). \qquad \square$$

The goal of the troubleshooting task is to find a troubleshooting strategy that minimizes the expected cost of repair among of all possible strategies.

2.6 Complexity of troubleshooting

The search for an optimal decision-theoretic troubleshooting strategy has appeared to be an NP-complete problem.

Theorem 2 *Given a troubleshooting problem with dependent actions, the single-fault assumption, and a constant $K \in \mathbb{R}^+$, determining if there exists a troubleshooting sequence s with ECR$(s) \leq K$ is an NP-complete problem.*

Proof: The idea of the proof is to reduce the problem to the Exact cover by 3-sets (see Sochorová & Vomlel (2000) for details). $\square$

Similar theorems may be proven for questions (even with independent actions) and dependent costs (even with independent actions and without questions).

Since we deal with an NP-complete problem we must resort to efficient heuristics to solve the problem within reasonable time. These heuristic methods are described in Section 4. First, however, we describe the models used for troubleshooting in the printing domain that motivated the development of the SACSO troubleshooting approach.

3 Printing system models

The SACSO printing diagnosis system consists of many separate Bayesian networks each modeling a printing error. If error conditions overlap and cannot easily be separated, they have to be represented in the same model. In printer systems there are the following types of error conditions:

- Dataflow models — these models cover problems where the customer does not get any output from the printer, or corrupted output from the printer when attempting to print. These errors can be caused by any of the components in the flow from application to printer that the print job passes through. Skaanning et al. (1998) have described these in detail.

- Error codes — these models handle all types of error codes that can appear on the control panel of the printer. Skaanning et al. (1998) have described this category in detail.

- Unexpected output — these models handle all categories of unexpected output that can occur on the printer, e.g., job not duplexed or spots, stripes, or banding on the paper.

- Miscellaneous — these models handle miscellaneous erroneous behavior of the printer not covered by the above three, such as noise from the printer engine, slow printing, problems with bi-directional communication, etc.

These error categories are related in the way that all error types can result in a general printer problem, and "Dataflow problems" can cause the three other error types.

Each of the SACSO models includes a *cause variable* that defines the probability distribution over the causes of the error condition. The causes are modeled as the states of this variable. All actions and questions that can be posed in the troubleshooting process are represented as children of the cause variable. An example is shown in Figure 3. The benefit of this *naïve Bayes* structure is that all actions and questions are independent given the causes. This can be exploited in the algorithms for finding the best next step, as shown in Section 4.

3.1 The unexpected-output models

The unexpected-output models represent all the situations where the customer does not get the expected output. This is usually due to settings not set correctly, or malfunctioning printer parts. Figure 4 shows an example Bayesian network model for an

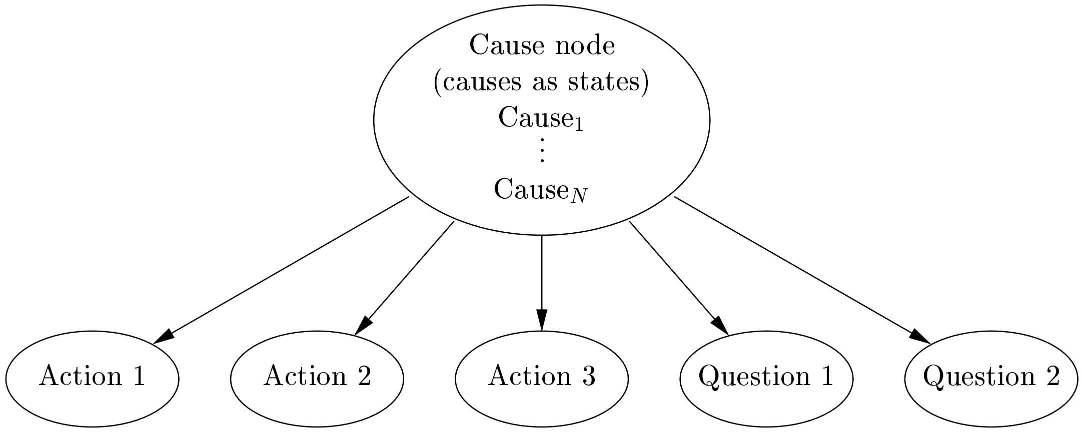

Figure 3: An example of the very simple Bayesian network structure used for troubleshooters.

unexpected output category, *Spots*. To enforce the single fault assumption the causes of this network are internally collapsed to a single node such as in Figure 3.

The customer may experience spots on the paper for some of the following reasons:

- The toner cartridge is malfunctioning either because it is defective or improperly seated.
- The used media has the wrong specifications.
- The environmental conditions of the printer may be out of specification, e.g., too humid, warm, etc.
- The transfer roller is malfunctioning either because it is defective, not seated correctly, or dirty.
- The power chord of the printer is not earth grounded.

3.2 The troubleshooting layer

The Bayesian network model pictured in Figure 4 is not sufficient for troubleshooting as it only contains information about the possible causes for the various problems with the printer. They contain no information on actions that can be used to resolve the problem at hand or gather information that can be used to speed up the troubleshooting. In this section, it will be described how variables representing information like this can be added to the structures presented in the previous sections.

We basically represent two types of troubleshooting steps; namely questions (including tests), which provide general information that can change the optimal sequence of troubleshooting steps, and actions, which can solve the problem.

In Figure 5, some troubleshooting actions and questions have been added to the model for the "HP MIO1 not ready" error code. The experts listed the actions and

Figure 4: An example of a Bayesian network model of the *Spots* category of unexpected output.

questions that they would usually perform when troubleshooting this error code over the telephone.

For each action it was determined which causes it could fix:

- Removing the network / IO cable can solve the problem if the network is the cause.
- Troubleshooting the entire dataflow can also solve the problem if the network is the cause. This action corresponds to the entire dataflow and all its troubleshooting steps.
- Waiting 5 minutes for initialization can solve the problem if the customer did not wait long enough.
- Cycling power can solve temporary problems and some intermittent. Even though intermittent problems are not really solved, this is the way it will look to the customer.

For each cause, fixable by an action, the printer experts have given a probability that the action would fix the cause, along with the cost of performing the action. The cost is based on 4 measures: the *time* it takes to perform the action, the *risk* of breaking something else while performing the action, the *money* involved in performing the action, and a potential *insult* by suggesting the action (e.g., check whether the power is on). These 4 factors are weighed and combined into a single figure.

3.3 An example run

Below, we have listed the steps generated by a troubleshooting tool called BATS Troubleshooter (see Section 4.3) in the presence of the error code "HP MIO1 not ready". Assuming that a defective MIO card is the cause of the problem, the troubleshooter will guide the customer through the following actions and questions.

1. Question: Did you wait 5 minutes for initialization? This question is given first to rule out the possibility that there is no problem at all. If the customer answers "no", he will be told to wait 5 minutes for proper initialization. As this does not solve the problem, the system continues.
2. Test: Move MIO card to another slot in the printer and try printing. This action tests whether there is a printer hardware problem with a broken MIO card slot. It does not solve the problem, and the system continues.
3. Repair action: Remove network / IO cable. This action can rule out a relatively likely cause (17%) with a very low cost (1 minute). It does not solve the problem, and the system continues.
4. Repair action: Ensure that the MIO card is supported by the printer. This will rule out situations where the customer is using a third party card or a card which is out of specifications. As the card is within specifications, the system will continue.

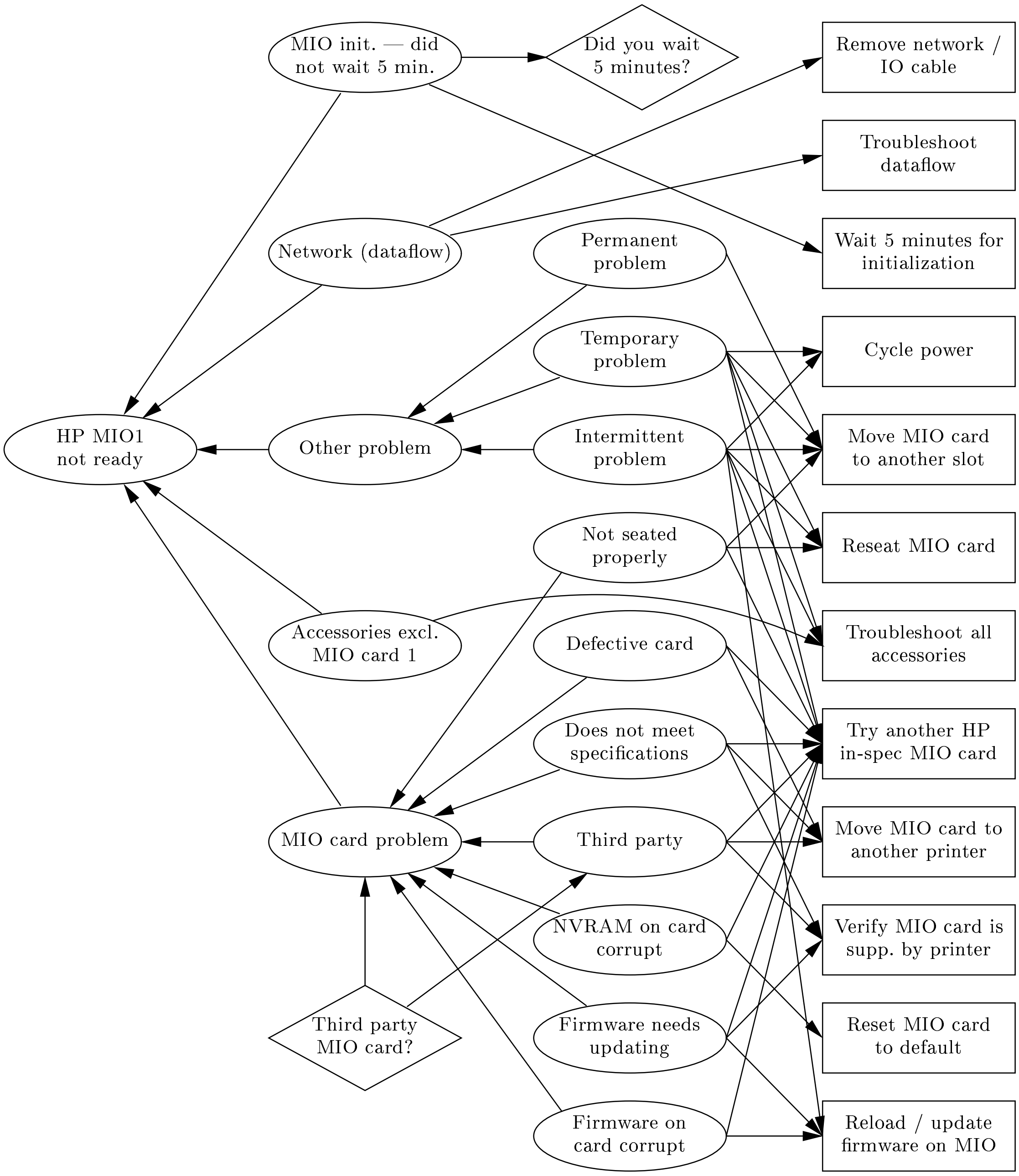

Figure 5: An error code model with added troubleshooting actions (rectangular shaped) and questions (diamond shaped).

5. Test:
   (a) Try another supported MIO card. This test can help ruling out one of the most likely causes, defective card (47%). It does solve the problem, but the system cannot say for sure whether it was because the original card was seated improperly, is a third party product, is out of specification, is defective, has corrupt NVRAM, or has corrupt or out of date firmware.
   (b) Reinsert the old card and test whether printing works now. This checks if the new card works because the old card was not seated properly. Since the old card is defective, it will obviously still not work.

The troubleshooter finally concludes that the MIO card is defective after ruling out the possibility of the card being seated improperly.

4 The SACSO troubleshooting approach

This section describes the SACSO approach to troubleshooting, and a tool implementing the approach is briefly described. We also compare the troubleshooting strategies obtained from the tool (and variants of it) with optimum troubleshooting strategies.

At any time in the troubleshooting process we wish to select the next step on the basis of the information gathered so far. Whenever a step has been performed and information from that step has been included, the same procedure for selecting the next step is repeated based on the updated information.

The basic idea behind selecting the next step is to compare the expected result of performing the repair action of highest efficiency with the expected result of asking a question (or performing a test). Our approach to evaluating the expected result of tests and questions is based on the following idea.

Assume, for example, that the fault is that the user has not installed a printer driver. Then the answer "no" to the question "Is there a printer driver installed?" will end the troubleshooting sequence. The rest will be instructions on how to get an appropriate driver and how to install it. Therefore, a question without any ability to fix the problem has a value. Entropy could be used as a measure of how focused the probability mass is. However, we have taken another approach in SACSO: if some answer q to question Q will identify the fault with almost certainty, then the value, V_Q, of asking Q is $P(Q = q)$. Mathematically, we calculate

$$P_Q(\varepsilon) = \max_i \max_q \frac{P(f_i \,|\, Q = q, \varepsilon) - P(f_i \,|\, \varepsilon)}{1 - P(f_i \,|\, \varepsilon)}.$$

The "good" answer is denoted q_G. If $P_Q(\varepsilon)$ exceeds a predefined threshold, V_Q is set to $P_Q(\varepsilon) \cdot P(q_G)$; otherwise it is set to zero. If there are several good answers, the corresponding values are added.

We extend Definition 3 and define the *current expected cost of repair* of a troubleshooting strategy s, given evidence ε compiled so far, as

$$ECR(s \mid \varepsilon) = \sum_{\ell \in \mathcal{L}(s)} P(\varepsilon_\ell \mid \varepsilon) \cdot t(\ell).$$

When s is clear from the context, we shall use $ECR(\varepsilon)$ as an abbreviation for $ECR(s \mid \varepsilon)$.

Let $\langle S_1, \ldots, S_n \rangle$ be the sequence of troubleshooting steps ordered according to the current efficiencies. As the assumptions in Proposition 2 are not met, it would be misleading to use Formula 5. Instead, we are forced to use Definition 1, and the calculation of ECR requires probability updating for each step in the sequence. Questions (and tests) are included in the sequence if their $P_Q(\varepsilon)$ is beyond a threshold close to 1 and if $\frac{P_Q(\varepsilon) \cdot P(q_G)}{C_Q}$ is maximal. When calculating ECR for a sequence containing a question, "the action has failed" means "$Q \neq q_G$". That is, $Q \neq q_G$ is inserted as evidence and used for the steps following Q.

We determine the troubleshooting step, A, of highest efficiency and calculate $\mathrm{ECR}(\varepsilon)$ as described above. Before actually performing A, we perform a two-step look-ahead analysis. Namely, we analyze whether a question should be asked.

For any question (and test) Q, we do the following. To determine the effect of asking Q, the expected cost of repair $\mathrm{ECR}(\varepsilon, Q = q)$ for each answer q is determined, and we calculate

$$\mathrm{ECR}_Q(\varepsilon) = C_Q + \sum_q \mathrm{ECR}(\varepsilon, Q = q) P(Q = q \mid \varepsilon). \tag{7}$$

If $\mathrm{ECR}_Q(\varepsilon) < \mathrm{ECR}(\varepsilon)$, the question Q should be asked. However, the comparison is biased. Unless Q is a question which might identify a cause, $\mathrm{ECR}(\varepsilon)$ does not take Q into consideration, and we have in fact analyzed the choice of asking Q now or never. Therefore, before it is decided to ask Q, it is analyzed whether it may be even better to ask Q after A has been performed:

$$\mathrm{ECR}_{A,Q}(\varepsilon) = C_A + \mathrm{ECR}_Q(\varepsilon, A = n) P(A = n \mid \varepsilon).$$

If $\mathrm{ECR}_{A,Q}(\varepsilon) < \mathrm{ECR}_Q(\varepsilon)$, the question is not asked, and if this holds for all Q with $\mathrm{ECR}_Q(\varepsilon) < \mathrm{ECR}(\varepsilon)$, A is performed. Note that the calculation of $\mathrm{ECR}_{A,Q}(\varepsilon)$ requires an entire new analysis. Notice also, that in case A fails, then a renewed analysis is performed.

4.1 Logical constraints and deferred actions

There are various constraints on the sequencing of the actions. For example, if the step "Reseat MIO Card" has been performed, the question "Is the MIO Card properly seated?" should not be asked. Some of these constraints are not consequences of the probabilities in the models. Therefore, the system keeps special account of these constraints, and it ensures that they are always met in the analysis of ECR and when proposing steps.

To improve the flexibility of the system, the user has the option of *deferring* a proposed action. A deferred action is still one of the options under consideration later unless the user requests for its removal.

4.2 Persistence and multiple faults

Often, a troubleshooting step changes the configuration of the system, and therefore the question of *persistence* is relevant: is the information acquired still valid? If not, and if the information is not updated, the system may go wild or into blind alleys. The printing system application was analyzed with respect to non-persistence, and it was concluded that this was not a problem. Actually, there are actions that change the configuration of the system. However, these actions either return the system to its original state upon failure, or modify components that will not be referred to and have an effect on the system later in the sequence.

The modeling and the sequencing method rely heavily on the single-fault assumption. If there are multiple faults, the proposed sequence will eventually fix them; perhaps at an unnecessarily high price. In particular, non-persistence may be a real problem in case of multiple faults, as each successful repair action definitely changes the configuration of the system, and maybe even eliminates several faults. So, after each successful repair action, one may be forced to discard all previous evidence before continuing the troubleshooting.

4.3 BATS Troubleshooter

In this section we briefly describe BATS[1] Troubleshooter, which implements the SACSO troubleshooting approach described above.

Figure 6 shows a screenshot of BATS Troubleshooter. The troubleshooter guides the user through a good troubleshooting sequence to resolve the error condition that he is currently experiencing. The graphical user interface allows the experienced user to track the computations of the algorithms for finding the best next step. The troubleshooter can suggest repair actions that may solve the problem, or questions about the printing system.

The user interface shows the currently suggested steps, and waits until the user provides the result to the step (whether an action solved the problem or not, or the answer to a question). The currently suggested error condition is light print — a common problem on printers. The problem of light print has both hardware and software causes, and some of the first troubleshooting steps selected by the diagnostic engine attempt to decide whether the cause is in the hardware or software section, e.g., "Is the printer configuration page printed light?".

The troubleshooter continuously displays a list of causes sorted wrt. their probabilities, a list of troubleshooting steps sorted wrt. their efficiencies, and a list of questions (and tests) sorted wrt. ECR_Q (see (7)).

The user interface of BATS Troubleshooter also supports more advanced features such as forcing certain steps to be asked immediately, going back and forward in the

[1]Bayesian Automated Troubleshooting System.

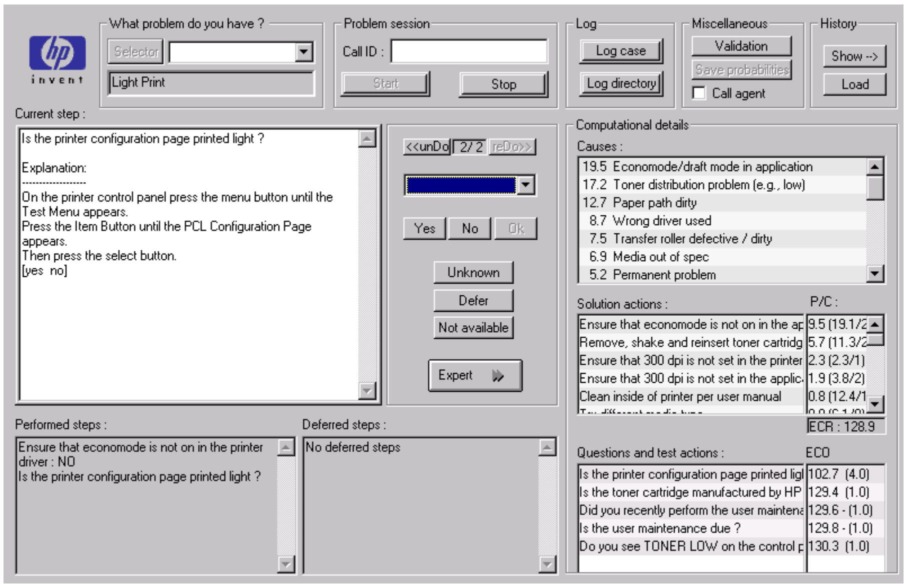

Figure 6: A screenshot of BATS Troubleshooter.

history, saving and loading restartable history files, logging XML format history files, etc. Skaanning et al. (2000) have described in detail the algorithms behind the selection of steps in BATS Troubleshooter.

4.4 Comparison of the SACSO approach with optimal strategies

As troubleshooting in general is NP-hard we have to use approximate methods. The space of possible troubleshooting strategies can be represented as a decision tree. There are basically two approaches for calculating approximate strategies: to perform a tree search using heuristics to prune the tree or to rely on a local computation whenever a new troubleshooting step has to be chosen. As the first approach requires a representation of the entire strategy (or frequent recalculations of it) we have chosen the latter approach in SACSO. In this section we compare the strategies provided by the SACSO method with the optimal strategies.

For comparison we have chosen a set of models of a size for which it was tractable to determine an optimal strategy. The optimal strategy was determined through a branch & bound algorithm. The branch & bound algorithm uses at each point a lower bound of the ECR for the remaining troubleshooting strategy.

Definition 4 Let $\mathcal{F}$ be the set of all possible causes of the problem and for every $F \in \mathcal{F}$ let the strategy s_F denote an optimal strategy given $F = \text{yes}$. We define a lower bound of the ECR as

$$\underline{\mathrm{ECR}} = \sum_{F \in \mathcal{F}} P(F = \text{yes}) \cdot \mathrm{ECR}(s_F). \qquad \square$$

Sochorová & Vomlel (2000) discuss in detail the properties of $\underline{\mathrm{ECR}}$. The computation of $\mathrm{ECR}(s_F)$ is usually quite easy. If it holds that the success probabilities for the various actions are independent and if we have independent costs, an optimal sequence is achieved by ordering the actions according to decreasing efficiency. Under all circumstances, for the models we are working with, the set of actions addressing the same fault is very small.

Our implementation of the branch & bound algorithm performs depth first search with pruning. Suppose that the algorithm gets to a node corresponding to evidence ε compiled so far, where $\mathrm{ECR}'(\varepsilon)$ — the lowest value of ECR from all subtrees passed through — is stored. Further, suppose that the step, S, under consideration has outcomes $s_1, \ldots, s_q, \ldots, s_r$ and that, for evidence $\varepsilon_i = \varepsilon \cup \{S = s_i\}$,

- the optimal value of $\mathrm{ECR}(\varepsilon_i)$ is already known for $i = 1, \ldots, q$ and
- the value of $\underline{\mathrm{ECR}}(\varepsilon_i)$ is computed for $i = q+1, \ldots, r$.

The pruning of subtrees corresponding to strategies starting with troubleshooting step S is performed as soon as we are sure that these strategies cannot be better than the current lowest one, i.e., when

$$\mathrm{ECR}'(\varepsilon) \leq C_S + \sum_{i=1}^{q} P(S = s_i \,|\, \varepsilon) \cdot \mathrm{ECR}(\varepsilon_i) + \sum_{i=q+1}^{r} P(S = s_i \,|\, \varepsilon) \cdot \underline{\mathrm{ECR}}(\varepsilon_i).$$

Since the function $\underline{\mathrm{ECR}}$ provides lower bounds of the optimal ECR, the optimal strategy cannot be missed.

4.5 Results

We have compared the strategies provided by the methods listed in Table 1. SACSO-A and SACSO-B differ on the criteria for selecting the next step, and SACSO is a combination of the two. The comparison is performed for 9 of the SACSO models for troubleshooting laser printers.

Table 2 summarizes the comparisons (details are provided by Vomlel (2000)). The last row of Table 2 summarizes the comparison as the average relative deviation from the optimal strategy. This shows that the troubleshooting strategies suggested by the SACSO approach are very close to optimal strategies, although the computational complexity of the SACSO approach is orders of magnitude lower than that of an optimal algorithm.

5 Validation

Validation of troubleshooters based on Bayesian networks poses a potential bottleneck. The system described here allows a number of sequences fulfilling various criteria from the troubleshooting models to be generated with the so-called *case generator*. These sequences can then be evaluated with the so-called *case evaluator*. If a sufficient number

Label	Approach
OPTIM	Optimal strategy minimizing ECR
SACSO	SACSO approach
SACSO-A	restricted SACSO approach where questions are selected based on P_Q only
SACSO-B	restricted SACSO approach where questions are selected based on ECR_Q only
P/C	The sequence of actions ordered according to step by step updated p/C-ratio

Table 1: Troubleshooting approaches.

# actions	# obs.	OPTIM	SACSO	SACSO-A	SACSO-B	P/C
6	2	433.24	442.39	444.54	442.39	444.54
9	3	129.21	129.21	129.21	129.21	155.10
11	3	106.20	112.35	113.36	108.07	116.80
12	3	38.38	38.42	38.42	40.01	43.05
13	4	124.32	124.37	298.09	125.56	300.85
14	4	115.41	115.86	232.05	115.86	236.58
9	9	70.67	75.03	119.28	77.67	121.10
16	5	161.38	162.25	286.75	162.25	286.75
10	10	250.45	253.31	352.31	256.96	479.96
Av. rel. dev. from opt.			1.81%	48.60%	2.51%	59.16%

Table 2: Comparison of values of ECR.

of these sequences are accepted, the model has an acceptable level of quality. If not, the model must be revised.

The validation method allows the generation of sequences in two different ways, (i) *random* sequences can be generated using the probabilities in the model, (ii) *special* sequences can be generated fulfilling various criteria such as sequences with the largest number of steps, sequences with the highest total cost, sequences ending with "call service", etc.

5.1 The case generator

Generation of random sequences is performed utilizing two diagnostic engines that are being executed in tandem, one with knowledge of the randomly chosen cause used to generate answers for steps (Engine 2 in Figure 7) and one with no knowledge used to suggest the sequence of steps (Engine 1).

Figure 7 illustrates the process followed to generate a sequence of random steps based on the probabilities of the model. In the left-hand side of Figure 7, the process of the diagnostic Engine 1 is shown, and the process of the diagnostic Engine 2 is shown in the right-hand side. The flow of control is illustrated by the arrows.

The case generator can also traverse the possible sequences and fetch the sequences with the highest number of steps or those with the highest total cost. Traversing all possible sequences may be infeasible, in which case the case generator can be stopped once a sufficient number of sequences have been traversed.

5.2 The case evaluator

For a quick overview, the case evaluator can provide a set of random sequences for each cause. When confronted with a sequence, the expert may accept it or discard it with a comment explaining what should be modified in the model. So far, modifications have to be performed manually, and it is an issue for further research to come up with efficient methods for automatic conservative refinement: how to change parameters without altering the accepted sequences.

If the random sets provided by the case generator are acceptable, the expert can start a more systematic evaluation by requesting "unfavorable" sequences. That may, for example, be lengthy sequences, costly sequences, sequences with high overhead or confusing sequences.

6 Further research

One type of tasks for further research involves relaxation of the assumptions listed in Section 2. Although the troubleshooting task is NP-complete under relaxed conditions, it is still important to find efficient heuristics which have good chances of providing close-to-optimal sequences. Certainly, diagnosing multiple faults is important, but there are other equally important tasks. For man-made devices one often meets the conditional cost problem: When fixing or inspecting a certain part one has dismantled the device

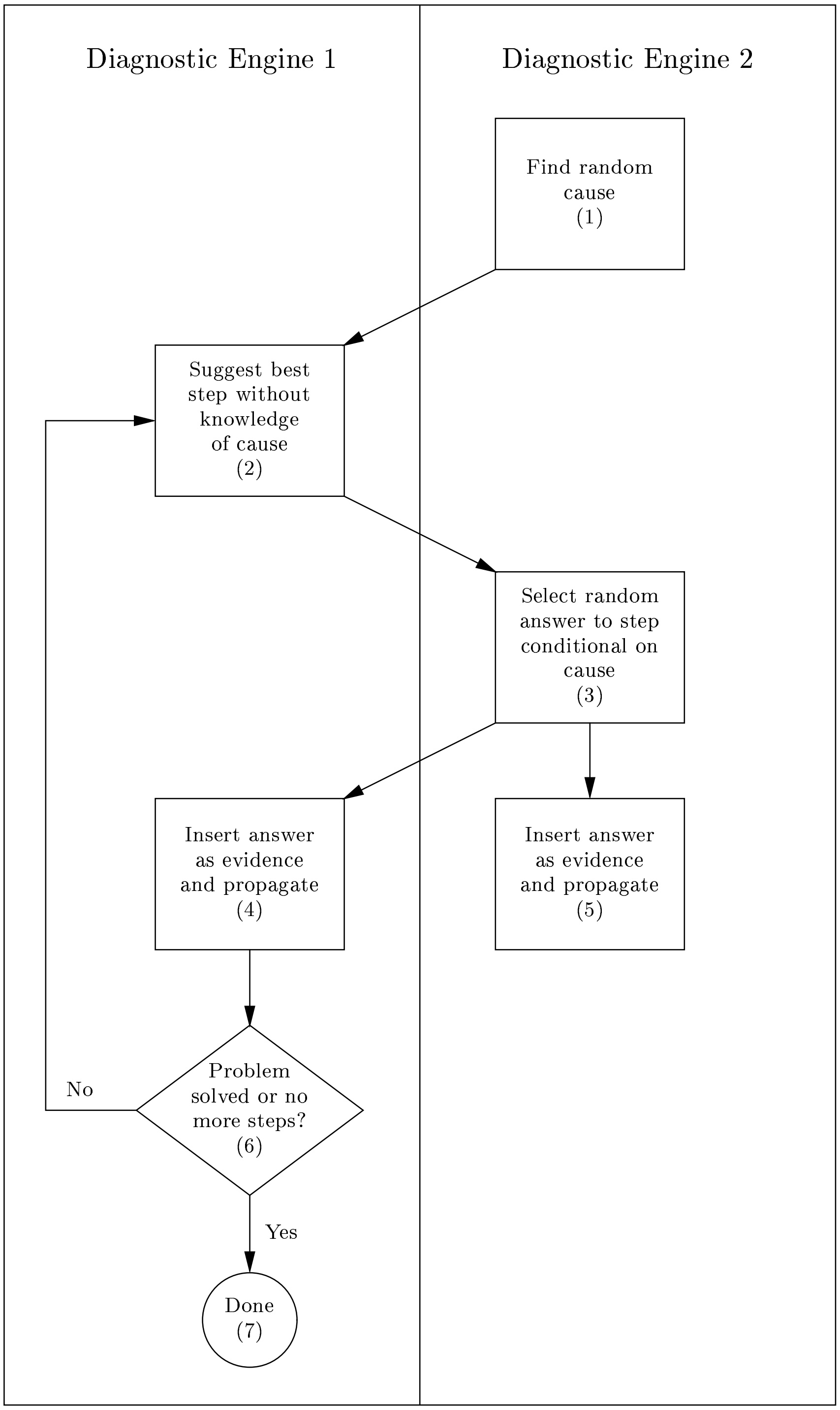

Figure 7: The flow of simulating random cases utilizing two diagnostic engines with and without knowledge of the true cause.

and before putting it together one may just as well perform other troubleshooting steps. Also, dependent actions (see Figure 1) is often seen.

Another type of research tasks has to do with quality of the model. Examples are conservative refinement (see Section 5), sensitivity analysis, learning, and adaptation. For the printer system it turned out that persistence was not a problem, but this does not hold in general: when a part of a system has been changed or reinstalled, how much of the previous evidence is then still valid?

Acknowledgements

We thank our co-workers on SACSO, in particular Olav Bangsø, Thomas Nielsen, Kristian G. Olesen, Lynn Parker, Paul Pelletier, and Lasse Rostrup-Jensen. The research was supported by the National Centre for IT Research through grant #87.2.

References

de Kleer, J. & Williams, B. (1987). Diagnosing multiple faults, *Artificial Intelligence* **32**: 311–319.

Heckerman, D., Breese, J. S. & Rommelse, K. (1995). Decision-theoretic troubleshooting, *Communications of the ACM* **38**(3): 49–56. Special issue on real-world applications on Bayesian networks.

Kalagnanam, J. & Henrion, M. (1990). A comparison of decision analysis and expert rules for sequential analysis, *in* P. Besnard & S. Hanks (eds), *Uncertainty in Artificial Intelligence 4*, North-Holland, New York, pp. 271–281.

Skaanning, C. (2000). A knowledge acquisition tool for Bayesian-network troubleshooters, *in* C. Boutilier & M. Goldszmidt (eds), *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, pp. 549–557.

Skaanning, C., Jensen, F. V. & Kjærulff, U. (2000). Printer troubleshooting using Bayesian networks, *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* .

Skaanning, C., Jensen, F. V., Kjærulff, U., Pelletier, P. & Rostrup-Jensen, L. (1998). Printing system diagnosis: A Bayesian network application, Workshop on Principles of Diagnosis, Cape God, Massachussets.

Sochorová, M. & Vomlel, J. (2000). Troubleshooting: NP-hardness and solution methods, *The Fifth Workshop on Uncertainty Processing WUPES'2000, Jindřichův Hradec, Czech Republic.*

Srinivas, S. (1995). A polynomial algorithm for computing the optimal repair strategy in a system with independent component failures, *in* P. Besnard & S. Hanks (eds), *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, pp. 515–522.

Vomlel, J. (2000). On quality of BATS troubleshooter and other approximative methods, *Technical report*, Department of Computer Science, Aalborg University, Denmark.

Author Biographies

Finn V. Jensen Professor in Computer Science at Aalborg University, Denmark. Ph.D. in Mathematical Logic, Warsaw University, Poland (1974). His scientific contributions have for the last ten years mainly been in connection to Bayesian networks and decision graphs. He is one of the founders of the Hugin method and the Hugin Company.

Uffe Kjærulff Associate Professor in Computer Science at Aalborg University, Denmark. Ph.D. in Computer Science, Aalborg University (1993). His research interest is reasoning under uncertainty. His scientific contributions relate mostly to methods for efficient inference in Bayesian networks.

Brian Kristiansen Research Assistant and Software Engineer at Hewlett-Packard Company. M.Sc. in Computer Science, Aalborg University, Denmark (1999). He has worked for Hewlett-Packard on the SACSO project since 1999.

Helge Langseth Research Assistant at the Department of Computer Science, Aalborg University, Denmark. Ph.D. student in mathematical statistics at the Norwegian University of Science and Technology.

Claus Skaanning Research Engineer at Hewlett-Packard Company. Ph.D. in Computer Science, Aalborg University, Denmark (1997). Worked for Hewlett-Packard on the SACSO project 1997-2000.

Jiří Vomlel Research Assistant at the Department of Computer Science of Aalborg University, Denmark. Ph.D. in Artificial Intelligence, Czech Technical University, Prague, the Czech Republic (2000). His research interest is reasoning under uncertainty.

Marta Vomlelová (Sochorova) Research assistant at Aalborg University, Denmark; M.Sc. in Artificial Intelligence, Charles University, Prague; Ph.D. student at University of Economics, Prague. Her research interest is probabilistic modeling.

Full postal address

Professor Finn V. Jensen
Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7
DK-9220 Aalborg
Denmark

Telephone: +45 9635 8080
Fax: +45 9815 9889
E-mail: fvj@cs.auc.dk