The SACSO System for Troubleshooting of Printing Systems

Finn V. Jensen
Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7C
DK-9220 Aalborg, DENMARK
fvj@cs.auc.dk

Claus Skaanning
Hewlett-Packard
Customer Support R&D
Fredrik Bajers Vej 7C
DK-9220 Aalborg, DENMARK
claus_skaanning@hp.com

Uffe Kjærulff
Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7C
DK-9220 Aalborg, DENMARK
uk@cs.auc.dk

Abstract

We report on the construction of a troubleshooting system for fixing faults in printing systems. The basic troubleshooting approach is outlined, concluding with a set of assumptions ensuring that a greedy approach will yield an optimal sequence of actions. The assumptions are weaker than the assumptions proposed by Heckerman, Breese & Rommelse (1995). The printing system domain 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. The validation process for the troubleshooter is reported.

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 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.

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). Particularly, 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 by Heckerman et al. (1995), taking advantage of the opportunity to perform many "propagations".

2 THE BASIC TROUBLESHOOTING TASK

Assume that we wish to troubleshoot a malfunctioning device. Assume that we have n possible actions $A_1, \ldots, A_n$. The outcome of an action can be y (the device was repaired), or n (the device was not repaired). Each action has two aspects, a *repair aspect* and an *observation aspect*. The repair aspect is represented by the probabilities $P(A_i = y \mid e)$, the probability that the device will be repaired given evidence, e. The observation aspect is represented by the probabilities $P(A_j = y \mid A_i = n, e)$, the probability that A_j will repair the device if A_i has failed. Each action, A_i, has a cost, $C_i(e)$, which may be dependent on the evidence. The observation aspect of an action may be further refined by extending the set of states allowing for a more elaborate description of action failures. In this section, we will assume the actions to have only the states y and n.

Let $S = A_1, \ldots, A_n$ be a repair sequence. That is, first A_1 is performed, and if this does not repair the device, then A_2 is performed, etc. We wish to come up with a repair sequence which minimizes the *expected cost of repair, ECR*: consider step i, and let e be the statement that all previous actions failed. The contribution to the expected cost of repair for step i is $P(e)C_i(e)$, and the contribution from step $i+1$ is $P(e, A_i = n)C_{i+1}(e, A_i = n)$. Through an easy calculation we see that if

$$C_i(e) + P(A_i = n \mid e)C_{i+1}(e, A_i = n) > \\ C_{i+1}(e) + P(A_{i+1} = n \mid e)C_i(e, A_{i+1} = n),$$

then the repair sequence with A_i and A_{i+1} swapped has a lower expected cost of repair.

We can conclude that for an optimal repair sequence, we have for all i

$$C_i(e) + P(A_i = n \mid e)C_{i+1}(e, A_i = n) \leq \\ C_{i+1}(e) + P(A_{i+1} = n \mid e)C_i(e, A_{i+1} = n). \quad (1)$$

2.1 EFFICIENCY OF ACTION

Though Formula 1 reduces the search space considerably, it neither gives a sufficient condition for a repair sequence to be optimal, nor does it yield an easy way of constructing an optimal repair sequence. If we assume that *the costs are independent of the actions performed*, Formula 1 can be rewritten to (cf. Kalagnanam & Henrion (1990))

$$\frac{P(A_i = y \mid e)}{C_i} \geq \frac{P(A_{i+1} = y \mid e)}{C_{i+1}}. \quad (2)$$

The term $\frac{P(A_i = y \mid e)}{C_i}$ is called *the efficiency* of action A_i given e, and the rule is, that *the actions are taken in a sequence of decreasing efficiency*. Still, Formula 2 is not a sufficient condition for an optimal repair sequence (see Figure 1).

In Figure 1 there are 4 possible causes for the 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. Assume that all actions have cost 1. Then A_2 has the highest efficiency, and if A_2 fails, then A_1 has higher efficiency than A_3. The sequence A_2, A_1, A_3 has ECR 1.50. However, the sequence A_3, A_1 has ECR 1.45.

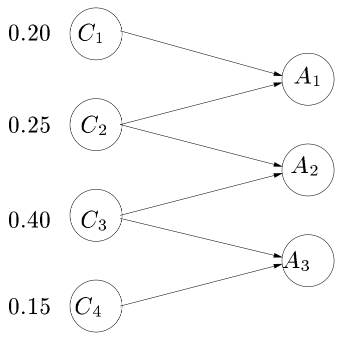

Figure 1: An example of dependent actions. The C's are causes for the device failing. The A-variables represent actions. An action will repair a parent, if faulty

2.2 CALL SERVICE

It is often the case that the troubleshooter has the option of calling outside assistance. This is usually costly. On the other hand, calling service will certainly solve the problem. Therefore, you can add a special action CS, which is always the last action in a repair sequence. The advantage of calling service is that you do not waste resources trying to fix a problem which you cannot fix yourself, but the disadvantage is that you might be able to fix the problem cheaper yourself. The cost of CS to consider is not the unknown price of fixing the device, but it is the possible overhead of having outsiders fix a problem which you could have fixed yourself.

Let C_S be the cost of CS, and let $A_1, \ldots, A_k, CS$ be a repair sequence ending with a service call. Then the *expected cost of repair* is

$$\begin{aligned} ECR(A_1, \ldots, A_k, CS) = \\ & C_1 + P(A_1 = n)C_2 + \ldots + \\ & P(A_1 = n, \ldots, A_i = n)C_{i+1} + \ldots + \\ & P(A_1 = n, \ldots, A_k = n)C_S. \quad (3) \end{aligned}$$

Now, consider the situation where A_k should be chosen. The question is whether it would be better to call service instead. Let e be the evidence gathered so far. Calculating the difference in ECR, we have

$$ECR(A_1, \ldots, A_k, CS) - ECR(A_1, \ldots, A_{k-1}, CS) =$$

$$P(e)(C_k + P(A_k = n \mid e)C_S) - P(e)C_S.$$

We see that service should be called if

$$\frac{1}{C_S} > \frac{P(A_k = y \mid e)}{C_k}.$$

2.3 ASSUMPTIONS FOR DECREASING EFFICIENCY

Let us analyse why the decreasing efficiency approach does not guarantee an optimal sequence. Let $A_1, \ldots, A_n$ 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

$$\frac{P(A_i = y \mid e)}{C_i} > \frac{P(A_j = y \mid e)}{C_j}.$$

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

$$\frac{P(A_i = y \mid e')}{C_i} < \frac{P(A_j = y \mid e')}{C_j},$$

where e and e' are evidence of the type: "the actions $A, \ldots, B$ have failed". We can infer

Property 1 *Repair sequence $A_1, \ldots, A_n$ is optimal if for all $i < j$ it holds that*

$$\frac{C_i}{C_j} \leq \frac{P(A_i = y \mid e)}{P(A_j = y \mid e)}, \qquad (4)$$

where e is any evidence of the type "actions $A, \ldots, B$ have failed" (excluding A_i and A_j).

Proposition 1 *The Property 1 holds under the following assumptions which are a modification of the assumptions formulated by Kalagnanam & Henrion (1990) and Heckerman et al. (1995).*

- *The device has n different faults $f_1, \ldots, f_n$ and $n+1$ different repair actions $A_1, \ldots, A_n, CS$.*
- *Exactly one of the faults is present.*
- *Each action has a specific probability of repair, $p_i = P(A_i = y \mid f_i)$, and $P(A_i = y \mid f_j) = 0$ for $i \neq j$.*
- *Action CS repairs with probability 1 any fault at the cost C_s.*
- *The cost C_i of a repair action does not depend on previous actions.*

Note that contrary to previous approaches, 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 $P(A_i = y \mid A_m = n)$ (for notational convenience we omit mentioning of the current evidence, e). Due to the single-fault assumption, we have

$$\begin{aligned}
P(A_i = y \mid A_m = n) & \\
&= \sum_k P(A_i = y, f_k \mid A_m = n) \\
&= P(A_i = y, f_i \mid A_m = n) \\
&= P(A_i = y \mid f_i, A_m = n) P(f_i \mid A_m = n) \\
&= \frac{P(A_i = y \mid f_i) P(A_m = n \mid f_i) P(f_i)}{P(A_m = n)} \\
&= \frac{P(A_i = y \mid f_i) P(f_i)}{P(A_m = n)} \\
&= \frac{P(A_i = y, f_i)}{P(A_m = n)} \\
&= \frac{P(A_i = y)}{P(A_m = n)}.
\end{aligned}$$

That is, $P(A_m = n)$ is a normalizing constant for the remaining actions, and Property 1 holds. □

As the order of the set of actions excluding CS will not be changed with new evidence, we can determine the sequence of them from the initial probabilities $p_j = P(A_j = y)$ alone, and the formula for expected cost of repair becomes

$$ECR = \sum_{i=1}^{n} C_i (1 - \sum_{j=1}^{i-1} p_j). \qquad (5)$$

However, this does not hold for CS. As $P(CS = y \mid e)$ always equals 1, we need to determine the smallest number i where

$$P(A_i = y \mid \text{all actions before } A_i \text{ have failed}) < \frac{C_i}{C_S}.$$

For this we can use the proof above recursively, and it is simple to determine when to stop and call service as well as to calculate the expected cost of repair for the resulting sequence.

2.4 VALUE OF IDENTIFYING THE FAULT

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, as an answer in this respect is only valuable if it actually ends the troubleshooting sequence, we have taken another approach in SACSO: if some answer q of the question Q will identify the fault with almost certainty, then the value of asking Q is $P(Q=q)$. Mathematically, we calculate

$$p_Q(e) = \max_i \max_q \frac{P(f_i \mid Q=q,e) - P(f_i \mid e)}{1-P(f_i \mid e)}.$$

The "good" answer is denoted q_G. If $p_Q(e)$ exceeds a threshold, the value of asking Q is set to $p_Q(e) \cdot P(q_G)$, otherwise it is set to zero. If there are several good answers, the corresponding values are added.

3 THE PRINTING SYSTEM MODELS

The SACSO printing diagnosis system consists of more than 200 separate models each representing a specific printing error called an *error condition* (for example "Light print"). Each error condition has a specific set of possible faults causing it. In the modelling, we assume the single-fault assumption to hold for each error condition, and we join the faults into states of a single variable. The justification for the single-fault assumption is that we assume that the system has been working properly so far, and suddenly a problem occurs. In that case, allowing for multiple faults introduces an unjustified complexity. As described in Section 4.2, the system can also handle multiple faults, but we do not claim it to be good at it.

As a model is only used when its error condition is present, the basic model reduces to one variable F with a prior probability distribution reflecting the probabilities of the various faults given the error condition.

Each model is extended with variables representing the various troubleshooting steps. Each step is represented as a child of F. There are several kinds of steps.

- *Repair actions* (example: "Reseat transfer roller"): actions which may fix the problem. The states of repair actions are y and n. A repair action may fix several faults (example: "Recycle power"), and therefore the assumptions in Proposition 1 do not hold.
- *Test actions* (example: "Direct the output to another printer"): configuration changes to test whether the problem disappears. The states of test actions are y and n. Resume to prior configuration after the action.
- *Symptom questions* (example: "Is the printer configuration page printed light?"): questions capable of identifying the fault. Symptom questions may have more than two states.
- *General questions* (example: "What type of driver is used?"): questions providing background information. General questions may have more than two states.

The acquisition of prior probabilities for the F-variables requires special care. As the numbers to specify can be very small, experts have difficulties in coming up with sufficiently precise estimates. Therefore, the faults were partitioned into subcauses and each subcause could furthermore be partitioned into subsubcauses (see Figure 2). Each subcause represents an identifiable component, and the experts give conditional probabilities of the type "Given error code A and given it is caused by subcause C_i, then the probability that we have fault f_j is x" or "Given error code A, then the probability that it is caused by a fault in subcause C_i is y".

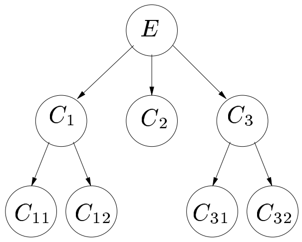

Figure 2: The error condition E may be explained by one of the subcauses C_1, C_2, C_3. Cause C_1 may be explained by one of the subcauses C_{11}, and C_{12}, and C_3 may be explained by one of C_{31} and C_{32}.

The calculation of the priors for F is then a matter of multiplying the estimated conditional probabilities.

To assess the probabilities of actions solving a particular fault f, the probability is dependent on whether the action is performed correctly and whether the requisites for performing the actions are fulfilled. That is, the expert is asked to provide $P(A = y \mid f, \text{correct}, \text{requisites})$ as well as $P(\text{correct})$ and $P(\text{requisites})$.

For each action, we elicited the cost factors *time, risk* (of breaking something else), *money*, and *insult*

("Check whether the parallel cable is plugged in" may insult an expert user). Time was elicited in minutes and money in dollars. Risk and insult are specified on a scale from 0 to 4. The elicited cost factors are combined linearly to form the overall cost of the action.

The resulting models are so simple that probability updating is very fast. Let M be a model activated by an error condition being present, and assume that we currently have the probability distribution $P(F)$ for the fault variable. Assume that the action A has failed, and let A be concerned with the fault f. That is, $P(A = y \mid f) = x > 0$, and $P(A = n \mid g) = 1$ for all $g \neq f$. Then

$$P(F \mid A = n) = \mu P(F) P(A = n \mid F),$$

where μ is a normalizing constant. As normalization can wait and $P(A = n \mid g) = 1$ for all $g \neq f$, we see that all we need is to multiply the number $P(f)$ with $1 - x$. Now, if we want to update the marginal probabilities for the remaining steps we can — for example for the question Q — do the following

$$\begin{aligned} & P(Q \mid A = n) \\ & \quad = \sum_F P(Q \mid F, A = n) P(F \mid A = n) \\ & \quad = \sum_F P(Q \mid F) P(F \mid A = n) \\ & \quad = \mu \left[\sum_F P(Q \mid F) P(F) - P(Q \mid f) P(f) + \right. \\ & \quad \quad \left. P(Q \mid f) P(A = n \mid f) \right] \\ & \quad = \mu (P(Q) - P(Q \mid f)(P(f) - (1 - x))). \end{aligned}$$

Again, a very simple operation which easily generalizes to actions concerned with several faults.

4 THE TROUBLESHOOTING APPROACH IN SACSO

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

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 performing a test action or to ask a question.

In the process we work with *the current expected cost of repair* $ECR(e)$. Let $S_1, \ldots, S_n$ be the set of troubleshooting steps ordered after the current efficiencies. As the assumptions in Proposition 1 are not met, it would be misleading to use Formula 5. Instead, we are forced to use Formula 3, and the calculation of ECR requires probability updating for each step in the sequence. Questions and test actions are included in the sequence if their $p_Q(e)$ is beyond a threshold close to 1, and if $\frac{P_Q(e) \cdot P(q_G)}{C_Q}$ 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 repair action A of highest efficiency and calculate $ECR(e)$ as described above. Before actually performing A, we analyse whether a question should be asked.

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

$$ECR_Q(e) = C_Q + \sum_q ECR(e, Q = q) P(Q = q \mid e).$$

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

$$ECR_Q(e \mid A) = C_A + ECR_Q(e, A = n) P(A = n \mid e).$$

If $ECR_Q(e \mid A) < ECR_Q(e)$, the question is not asked, and if this holds for all Q with $ECR_Q(e) < ECR(e)$, A is performed. Note that the calculation of $ECR_Q(e \mid A)$ 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 "Install a new driver" has been performed, the question "Do you have a driver installed?" 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

Usuallly 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 analysed 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 modelling 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.

5 STRATEGY TREES AND VALIDATION

In principle, the end result of a troubleshooter project is a set of strategy trees: for each error condition E you have a tree with E as root, and the remaining nodes represent actions and questions; the tree branches at nodes representing questions. It is a matter of space and time complexity whether a strategy tree is represented directly, or whether it is represented as a model with attached inference methods. In the SACSO project, almost all strategy trees were sufficiently small to be represented directly.

This does not mean that the models are inactive on-line. If the "defer" option is used, that is, if the user wishes not to perform a proposed step, then the precompiled strategy tree is not relevant anymore. As the "defer" option can be used at any time in the process, it is inconvenient to have precompiled strategy trees for all possible scenarios including the ones with "defer".

The models are also used for maintenance. This can be illustrated by the way the models were used for validation. After the construction of the models, the strategy trees for the models were constructed. Still the strategy trees had too many paths for manual inspection. Instead, a validation module picked certain paths of particular interest for inspection ("defer" was excluded from the validation). They were

1. **Common faults**. The module provides the n most common faults, and it provides typical troubleshooting sequences for each of them.
2. **Critical faults**. Some faults are critical in the sense that, if they are not corrected immediately, serious damage will occur later. The expert provides a list of the critical faults, and the module checks that they are not missed no matter the answers given — provided that the answers are consistent with the fault. It is up to the expert to evaluate whether the faults are spotted with reasonable effort.
3. **Lengthy sequences**. Some troubleshooting sequences may become very long. The module lists the most lengthy sequences.
4. **Costly sequences**. The module lists the most costly sequences. The expert checks that the costs are justified.
5. **Call service**. The module lists all sequences where the system quickly gives up and recommends to call service.
6. **Sequences with high overhead**. The module lists the sequences with high costs compared to the cost of direct repair (had you known the fault).

The result of validation is a list of accepted sequences and another list of sequences which are only accepted up to a certain step, and at this step a particular step is preferred to the one suggested by the system.

5.1 CONSERVATIVE REFINEMENT

Formally, the situation after validation can be described as follows. Let T be a strategy tree for an error condition. In T, certain paths from the root have been accepted. Actually, the accepted part of T is a connected subtree T_a containing the root. Let A be a node just outside T_a (see Figure 3).

Assume that the possible actions at this place in addition to A are $B_1, \dots, B_m$, and assume that an expert for this situation prefers action B_i to action A. Let the information acquired at the point be e. The task now is to change the parameters of the model (probabilities and costs) such that B_i is selected rather than A. However, this must neither change T_a nor any other accepted subtrees in the system. This is called *conservative refinement*.

For the SACSO project, the experience was that when the experts disagreed with the system, it was mainly due to a modelling error (an action not incorporated, a missing action constraint, etc.). It also happened that the experts got convinced that the system's suggestion was better than their own. In rare cases, they

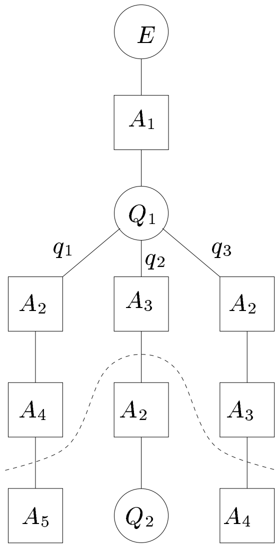

Figure 3: Part of a strategy tree. The dashed curve indicates the border of acceptance

had to adjust some parameters, and it was rather easy for them to ensure that the adjustments were conservative. Therefore, we did not put much effort into constructing computer aided systems for conservative refinement.

Furthermore, none of the disagreements between expert and system seemed to be due to a suboptimal method. For some of the models, an exhaustive search through the corresponding decision tree has been performed, and it showed only small differences in ECR between the optimal sequence and the sequence provided by the system.

6 CONCLUSIONS AND FUTURE RESEARCH

The SACSO collaboration was initiated a couple of years ago and has now resulted in fully functional troubleshooting systems for four HP laser printers. Each system consists of a couple of hundred naïve Bayes models with each model covering a specific error condition. The models are executed via a generic runtime module, called BATS Troubleshooter, which presents the troubleshooting steps and provides facilities for e.g. deferring actions, choosing alternative actions, and entering configuration specific information through an easy-to-use graphical user interface. Internal reviews of the systems have generated a great deal of enthusiasm, both at executive and call agent levels. The systems are currently undergoing critical validation by domain experts, and we expect plans for deployment to be finalized in the near future.

The knowledge acquisition process is supported by a special purpose tool, called BATS Author, which is described in detail in a forthcoming paper, Skaanning (2000). Experience has shown that domain experts with no previous exposure to Bayesian networks are able to build troubleshooting models after just a few days (or even hours) of training with this tool.

The SACSO activities have involved an application oriented as well as a research oriented path. In the latter, we have dealt with a variety of issues supporting the troubleshooting process. Some of these, which are described below, are still open-ended and need further research.

The troubleshooting environment provided by BATS Troubleshooter does not currently support learning/adaptation of the probability and cost parameters of the models. Some preliminary considerations have led to the conclusion that the EM algorithm seems to be well suited for learning the probability parameters, although the data only asymptotically seem to meet the missing-at-random requirement. Learning of cost parameters, on the other hand, seems much more difficult. When the user defers an action or forces one to be performed next — which was not suggested by the system — it might be an indication that the cost parameters should be adjusted. However, lacking knowledge of the user's motivation for deferring or forcing an action, inconsistencies between the user's subjective probabilities and the more objective ones of the system, and lacking information about the user's profile (level of expertise, etc.) make it very hard, if not impossible, to formulate a sound theoretical basis for cost learning.

Although validation of the SACSO models so far has not called for a sophisticated method for conservative refinement, we foresee a need for such a method. With cost learning being so difficult, it seems plausible to base a method for conservative refinement on modification of cost parameters.

The cost of an action, A, often depends on the actions which precede A; for example, the cost of performing action "Reseat paper tray" is lower if the preceding action was "Recycle power" than if it was "Check setting x in application y on your PC", as, in the first case, the user saves the overhead of walking to the location of the printer. Some preliminary thoughts as to how to modify the troubleshooting algorithm have been made, but it has not been implemented in BATS

Troubleshooter.

Also, some preliminary investigations have been made as to how the robustness of a model can be assessed. Having provided an initial set of model parameters, a domain expert might wish to have the most influential parameters highlighted and maybe call in additional expert opinions to reduce the uncertainty in these parameter estimates. Our investigations have taken their outset in algorithms for sensitivity analysis as described by Kjærulff & van der Gaag (2000).

As a final issue for future research, a formal analysis of the complexity of the troubleshooting task under different assumptions would be of interest. That is, under which combination of assumptions of a single fault, dependent actions, conditional costs, etc. is the troubleshooting task NP-hard?

Acknowledgements

We are grateful to our co-workers in SACSO: Lasse Rostrup-Jensen, Paul Pelletier, and Lynn Parker for modelling and validation, Pierre-Henri Wuillemin and Olav Bangsø for competent programming and valuable theoretical discussions, and Janice Bogorad for inspiring sparring.

The University involvement in SACSO is funded by the Danish National Centre for IT research, Project no. 87.

References

Heckerman, D., Breese, J. S. & Rommelse, K. (1995). Decision-theoretic troubleshooting, *Communications of the ACM* **38**(3): 49–57. 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.

Kjærulff, U. & van der Gaag, L. C. (2000). Making sensitivity analysis computationally efficient, Submitted to UAI 2000.

Skaanning, C. (2000). A knowledge acquisition tool for bayesian-network troubleshooters, Submitted to UAI 2000.

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, San Francisco, pp. 515–522.