Heuristics for two extensions of basic troubleshooting

Helge Langseth Finn Verner Jensen
Department of Computer Science
Aalborg University
email: {hl, fvj}@cs.auc.dk

Abstract. This paper describes efficient probabilistic troubleshooting of electro-mechanical devices. We relax two of the standard requirements made to ensure tractability of this task, by allowing cost structures to depend on the sequence of actions performed so far (conditional cost) as well as troubleshooting actions to remedy intersecting sets of faults (dependent actions). Heuristics for these two extensions of basic troubleshooting are proposed and evaluated through empirical studies.

1 Introduction

Assume you are confronted with a malfunctioning device, and you have a set of troubleshooting steps to perform. Some steps are *actions* which may or may not fix the problem. We say that these steps have a *repair aspect*. Other steps are *questions* which cannot fix the problem, but may give indications of the causes of the problem; these have an *observation aspect*. Some operations have both a repair aspect and an observation aspect, but these are not considered in this paper. All steps have a cost in terms of e.g. money or time. The task is to find the cheapest strategy for sequencing the troubleshooting steps, such that the device is eventually repaired.

To formalize, the system is in some fault state F. The troubleshooter (TS) system has a list of N possible actions to choose from to remedy the fault, denoted by A_i, $i = 1, \ldots, N$. When an action is performed, the system will get to know whether the action solved the problem ($A_i = y$) or failed ($A_i = n$). There are also M predefined questions Q_j, $j = 1, \ldots, M$ that may be posed. A TS-*step* is a step in a TS-sequence, either an action or a question. To each TS-step B_i the associated cost is denoted by C_i.

Decision-theoretic troubleshooting was studied in [4]. It was extended to the context of Bayesian networks in [1], where a framework for suggesting sequences of questions, actions, and configuration changes is provided. By calculating a local efficiency of the possible repair actions and continuously choosing the one of highest efficiency, a repair sequence is established. A number of assumptions have to be made to ensure optimality of the found sequence:

Single fault: Exactly one fault is present in the system. If the fault is identified the TS-session is ended.

Perfect repair: Actions solve their associated faults with certainty.

No questions: Each TS-step has a repair aspect, but no observation aspect.

Independent actions: If there exists a pair of actions (A_i, A_j) such that A_i can remedy at least one of the faults that A_j handles we say that the model has *dependent actions*. If no such pair exists the domain has independent actions.

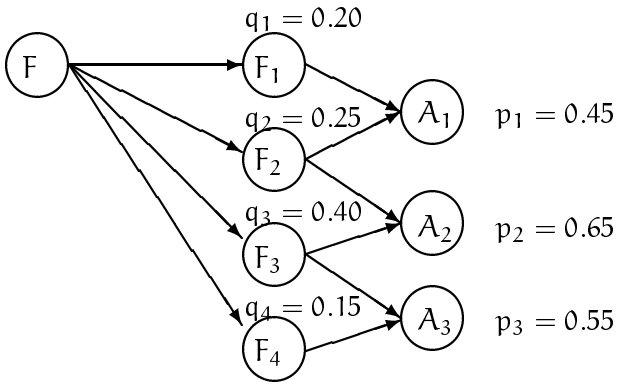


Figure 1: A simple example of a Bayesian troubleshooter. The values of $p_i = P(A_i = y)$ and $q_j = P(F_j = y)$ are indicated in the figure.

Fixed cost: If the costs of the TS-steps do not depend on the previous TS-steps the domain has fixed costs. Otherwise, we say the domain has *conditional cost*.

Given these assumptions the method in [1] finds the optimal sequence of actions. With respect to questions, a myopic one-step lookahead is suggested, but optimality is then no longer assured.

In our work we have created a troubleshooter system for a printer domain. A printing system consists of several components: the application where the printing command is sent from, the printer driver, the network connection, the server controlling the printer, the printer itself, etc. In this domain the single fault assumption seems reasonable, so our work has focused on relaxing the other assumptions. The next section will briefly give some results from basic troubleshooting. We will then consider heuristic methods for troubleshooting domains with conditional cost and dependent actions. Methods for handling questions and non-perfect repairs are presented in [3].

2 Basic troubleshooting

This section will give some basic definitions and results from the theory of troubleshooting systems. Proofs and further discussion can be found in e.g. [3]. We use the framework of *Bayesian networks* [2] to model the troubleshooter domain; a simple example taken from [3] is given in Figure 1. In the example there are 4 possible fault states named $f_1, \ldots, f_4$. These coincides with the state-space of the random variable F. The children of F, F_i, $i = 1, \ldots, 4$, are deterministic indicator variables that take the value $F_i = y$ if $F = f_i$ and $F_i = n$ otherwise. The action A_j is a child of exactly those faults that it can remedy, e.g., A_2 can remedy faults f_2 and f_3. An action A_j can only be successful ($A_j = y$) if one of its parents is in the y-state. The prior distribution over the states of F reflects the relative frequencies of each of the four faults. The probabilities for action A_i to solve the problem given fault $F = f_j$ indicates a combination of the probability that the action is performed correctly, and how well suited it is to solve the problem at hand. In this simple example all repairs are perfect, and the probability $P(A_i = y)$ can therefore be calculated by simple summation.

We will assume that the TS-model contains *no questions* when we state the results both in this section, and in the remaining of the paper. Troubleshooting with questions has been studied in e.g. [1, 3].

The goal of a TS-system is to provide a "good" troubleshooting sequence (TSS). Formally, a sequence $S = \langle A_1, \ldots, A_N \rangle$ of actions is called a *troubleshooting sequence* if it describes the process of repeatedly performing the next action in the sequence until an action succeeds ($A_k = y$) or until the last action has been performed. How "good" a TSS is, is judged by the *expected cost of repair*, ECR. The ECR of a TSS is expected cost of performing the TSS. A TSS is said to be *optimal* if it achieves the minimum ECR of all TSSs. Let $\boldsymbol{\epsilon}$ denote arbitrary evidence collected so far during troubleshooting. If a more specific definition of the collected evidence is needed, $\boldsymbol{e}_j$ denotes the evidence that the first j actions in the sequence $\langle A_1, \ldots, A_N \rangle$ have all failed, $\boldsymbol{e}_j = \{A_i = n \mid i = 1, \ldots, j\}$. $p_i = P(A_i = y \mid \boldsymbol{\epsilon})$ is the probability that action A_i solves the problem given the evidence $\boldsymbol{\epsilon}$. $\boldsymbol{\epsilon}$ is kept implicit in this definition of p_i as the evidence will be clear from the context. The *efficiency* of an action A_i, $ef(A_i \mid \boldsymbol{\epsilon})$, is defined as

$$ef(A_i \mid \boldsymbol{\epsilon}) = \frac{P(A_i = y \mid \boldsymbol{\epsilon})}{C_i} \tag{1}$$

It is easy to verify that if $S = \langle A_1, \ldots, A_n \rangle$ is an optimal TSS of repair actions with fixed cost, then it must hold that $ef(A_i \mid \boldsymbol{e}_{i-1}) \geq ef(A_{i+1} \mid \boldsymbol{e}_{i-1})$. As shown by Theorem 1 below a greedy search based on the efficiency of the actions results in an optimal TSS given the assumptions in Proposition 1.

Proposition 1 *If the following holds*

1. *The device has* N *faults* $f_1, \ldots, f_N$ *and* N *actions* $A_1, \ldots, A_N$.
2. *Exactly one of the faults is present.*
3. *Each action has a specific probability of repair,* $P(A_i = y \mid F = f_i)$*, and* $P(A_i = y \mid F = f_j) = 0$ *for* $i \neq j$.
4. *The cost* C_i *of an action does not depend on previous actions.*

If $ef(A_j) \leq ef(A_i)$ *then* $ef(A_j \mid \boldsymbol{\epsilon}) \leq ef(A_i \mid \boldsymbol{\epsilon})$*, where* $\boldsymbol{\epsilon}$ *is any evidence of the type "Actions* $A_k, \ldots, A_l$ *have failed" (excluding* A_i *and* A_j*).*

Note that we do not assume perfect repair in this proposition.

Theorem 1 *Let* $S = \langle A_1, \ldots, A_N \rangle$ *be a repair sequence for a troubleshooting problem fulfilling the conditions in Proposition 1. Assume that* S *is ordered according to decreasing initial efficiencies* $ef(A_i \mid \boldsymbol{e}_0)$*. Then* S *is an optimal repair sequence, and the* ECR *can be calculated as*

$$ECR(S) = \sum_{i=1}^{N} \left(1 - \sum_{j=1}^{i-1} p_j \right) C_i \tag{2}$$

Thus, the conditions in Proposition 1 are sufficient to ensure the existence of a polynomial algorithm to find the optimal TSS. If relaxed, however, the problem is intractable, [6]:

Proposition 2 *If we relax any of the following requirements*

- *No question*
- *Fixed costs*

- *Independent actions*

the problem of finding an optimal TSS *becomes* NP*-hard.*

The focus of the remainder of this paper is to generate reasonable approximations to the optimal TSS in a system with conditional costs or dependent actions. A similar study of troubleshooting with questions is to be found in [3].

3 Conditional cost

This section examines a TS-model fulfilling all conditions in Proposition 1 except for Requirement 4. Let $\boldsymbol{\pi} = \left(\pi_{(1)}, \pi_{(2)}, \ldots, \pi_{(N)}\right)$ be a permutation of the numbers $1, \ldots, N$, and assume that the TSS to be evaluated is given by $S = \langle A_{\pi(1)}, \ldots, A_{\pi(N)}\rangle$. By *conditional cost* we mean that the cost of an action depends on the sequence of actions performed so far, that is, the user can change the cost $C_{\pi_{(i)}}$ by performing actions in a given sequence before doing $A_{\pi(i)}$. Thus, the cost is a function of the history of performed actions, $C_{\pi_{(i)}} = f_{\pi_{(i)}}\left(\pi_{(1)}, \ldots, \pi_{(i-1)}\right)$ for some known function $f_{\pi_{(i)}}(\cdot)$. We simplify this general problem by enforcing a Markovian property on the cost function, so that $f_{\pi_{(i)}}(\pi_{(1)}, \ldots, \pi_{(i-1)}) = f_{\pi_{(i)}}(\pi_{(i-1)})$. We will think of this as a situation where a lower cost on a set of actions can be obtained by performing some kind of initial action first. A simple example from the printer domain is a man sitting in his office at the $13th$ floor printing to a printer located in the basement. If the printer is faulty, he will go to the basement and try to fix the problem. Once there, it seems reasonable to perform as many operations at the printer's site as possible to make sure that the printer is repaired before he leaves; otherwise he would have to take the long journey once more. Hence, it is cheaper to do an action that must be performed at the printer's site if he is already there, i.e., if the previous action also was to be performed in the basement.

It is assumed that an action can be broken into three distinct parts, the initial work, the problem-solving work and the finalization work. First, some initial work is conducted. This makes the user able to perform the "real" action, which is the second part. The third part amounts to finalizing the action, i.e., reverting the initial work. The sum of the costs of these parts is the total cost of the action. The cost of the initial work is C_0^i, the cost of the finalization work is C_∞^i and the cost of the "real work" is $\gamma_i =_{\text{def}} C_i - C_0^i - C_\infty^i$. For simplicity, the initial and finalization costs are denoted C_0 and C_∞ if it is not ambiguous. We assume that initial and finalization work can never solve the problem at hand. The sum $C_0^i + C_\infty^i$ is named the *enabling cost of action* A_i since taking on that cost enables us to perform the action's problem-solving work. In the example above, the initial work of the action "Cycle power on printer" is to go to the printer. The problem-solving part is to push the cycle power button, and the finalization work is to go back to the office.

Actions that have the same enabling work will be grouped together, and we will call such a group of actions a *cluster*. $C_0 + C_\infty$ is the enabling cost of all the actions in the cluster. As soon as this cost is taken, access to all actions in the cluster is gained. We assume that an action is a member of at most one cluster, and we refer to the cluster of action A_i as cluster(A_i). We also impose a "flat" internal structure upon the clusters, meaning that a cluster is not allowed to contain other clusters. An action that is not clustered with any other actions is called an *atomic action*.

The "cluster-framework" essentially means that the cost $C_{\pi_{(i)}}$ of action $A_{\pi_{(i)}}$ can be expressed as

$$C_{\pi_{(i)}} = \begin{cases} \gamma_{\pi_{(i)}} & \text{if cluster}(A_{\pi_{(i-1)}}) = \text{cluster}(A_{\pi_{(i)}}) \\ C_0^{\pi_{(i)}} + \gamma_{\pi_{(i)}} + C_\infty^{\pi_{(i)}} & \text{otherwise} \end{cases} \quad (3)$$

This type of conditional cost problems comes in two flavors. The most delicate situation is when the user does not have any information about the system's state while performing actions located "inside a cluster", that is, we are not able to check whether the problem is solved without performing the cluster's finalization work first. This is called TS *without inside information.* If the system's state can be checked at no cost at any time during the TS session it is called TS *with inside information.* In the example, it is not known if the problem is solved while the person is in the basement (i.e. "inside the cluster") since the printing system's condition should be tested by re-sending the print-job from the computer at the operator's office. If, however, he is connected by mobile phone to a colleague sitting by his computer, print-jobs can be sent regularly, and we have a situation *with* inside information.

To implement conditional cost calculations one must consider how to find the actions that can be grouped together. Then it must be determined when to take the initialization cost to access the actions within a cluster, and decided when to leave the cluster. This last step is important since it may be non-optimal to perform very expensive low-probability actions inside a cluster even though the enabling cost has been taken. The first part is the simplest one, as all actions with the same enabling work are considered to make up one cluster. (All operations performed at the printer site are clustered in the example.) The two other tasks will be considered in the remainder of this section.

3.1 Calculating the cluster's efficiency

To allow for the simple greedy approach based on Theorem 1 to be applied, formulas for the *efficiency of a cluster* must be established in the same way as (1) defines the efficiency of an atomic action. When the efficiency is found, we assume we can treat the cluster of actions as every other action, and decide to enter the cluster whenever that efficiency is the best one not yet performed. If, e.g.,

$$ef(A_i \mid \epsilon) \geq ef(\text{Cluster} \mid \epsilon) \geq ef(A_j \mid \epsilon) \quad (4)$$

we enter the cluster after performing action A_i. As soon as we decide to leave the cluster, the plan is to continue with action A_j.

For simplicity of notation we will assume that we have labeled the r actions sharing the same initial and finalization work as $A_1, \ldots, A_r$, and that we have ordered them so that $\frac{p_i}{\gamma_i} \geq \frac{p_j}{\gamma_j}$ for $i < j$. To give the cluster its correct position in the TSS defined by (4), we assume we should maximize the cluster's potential efficiency, i.e. define

$$ef(\text{Cluster} \mid \epsilon) \stackrel{\text{def}}{=} \max_k \frac{\sum_{j=1}^k P(A_i = y|\epsilon)}{C_0 + \sum_{j=1}^k \gamma_j + C_\infty} = \frac{1}{1 - P(\epsilon)} \cdot \max_k \frac{\sum_{j=1}^k P(A_i = y)}{C_0 + \sum_{j=1}^k \gamma_j + C_\infty} \quad (5)$$

The set of actions $\{A_1, \ldots, A_k\}$ that defines the cluster's efficiency is independent of the evidence ϵ. Let $\mathcal{J}_0$ denote this set of actions. Not all the actions in a cluster will be in $\mathcal{J}_0$; the cluster will be partitioned into sets $\mathcal{J}_0$, $\mathcal{J}_1$, $\mathcal{J}_2$, ... $\mathcal{J}_0$ is the set maximizing (5), $\mathcal{J}_1$ consists of the "best" actions that remain in the cluster when the actions in $\mathcal{J}_0$ have been performed, and so on. $ef(\text{Cluster} \mid \epsilon)$ as calculated by (5) is thus actually

the efficiency of the "best" set, $\mathsf{ef}(\mathcal{J}_0 \mid \epsilon)$; the efficiencies of the other partitions are calculated similarly. At any given time the system only needs to keep track of the next time to enter a cluster. Hence, in practice, only $\mathsf{ef}(\mathcal{J}_0 \mid \epsilon)$ will be calculated initially. When the actions in $\mathcal{J}_0$ have been performed, the efficiency of the cluster with the remaining actions must be calculated.

3.2 Pruning the selected set

The method outlined above is based on the assumption that the members of the cluster should be chosen to maximize the cluster's potential efficiency. This is in general only an approximation, and this section will consider how to adapt the method to better handle the special case of TS without inside information.

When the TS system is inside a cluster, it is necessary to be able to compare the problem-solving actions available within the cluster to the cluster's finalization work. The problem may have been solved without the TS-system knowing it, and continuing troubleshooting is non-optimal in this case. To gain information about the system's state, the finalization work of the cluster must be performed. Hence, lacking inside information gives a motivation for leaving the cluster earlier than what would have been assumed to be optimal with inside information. As soon as the TS is "outside the cluster", the system's state is acquired at no extra cost, and the TS-session may be ended. If the problem has not been solved, the efficiency of the remaining actions in the cluster is to be calculated so that the cluster can be entered again at the appropriate time. Leaving a cluster prematurely will give the return to the cluster a high efficiency, and it may therefore force the TS back into the cluster again. The result is that if the TS is too eager to leave the cluster, it may have to take on additional enabling costs.

In this paper we simplify these considerations by assuming that we can treat the lacking inside information in the same way as questions are handled in standard troubleshooting (see [3] for a description). Assume the cluster is left after performing the set of actions $\{\mathsf{A}_1, \ \ldots, \ \mathsf{A}_q\} \subset \mathcal{J}_0$. The TS should in principle recalculate the cluster's efficiency by (5) and, according to (4), continue with the cluster or atomic action ranked the highest. In our simplification, we assume that the TS will always return to the cluster if the problem has not been remedied. Furthermore, it is assumed that the set of actions representing the cluster is not recalculated. As the TS continues, the cluster will therefore consist of the same actions $\mathcal{J}_0 \setminus \{\mathsf{A}_1, \ \ldots, \ \mathsf{A}_q\}$ as before the cluster was left. Hence, the gain of leaving the cluster is just the information about the system's state. If the TS session is to continue, this will be from the same position as before this information was acquired. To summarize, we consider the following options:

1. Go outside the cluster to check if the problem is solved. Then continue troubleshooting inside the same cluster.

2. Do the best step inside the cluster, then check the system's state. Continue by entering the cluster again.

3. Do not leave the cluster before all actions in $\mathcal{J}_0$ are performed.

By plain algebraic manipulation of the definition of ECR it is easy to show that one should leave the cluster whenever

$$\frac{\mathsf{p}_{\,\mathrm{Solved}}}{\mathsf{C}_0 + \sum_\kappa \gamma_\kappa + \mathsf{C}_\infty} \geq \frac{\mathsf{p}_\ell}{\gamma_\ell} \tag{6}$$

where ℓ is the next best action to do in $\mathcal{J}_0$, and the sum over κ is taken over the actions in $\mathcal{J}_0$ that are not yet performed, excluding ℓ. p_{Solved} is given by the probability of having solved the problem without knowing it,

$$p_{\text{Solved}} = \sum_{\text{Since last check}} p_i$$

When there is only one action left inside the cluster, equation (6) is no longer valid. In this case we compare alternatives 1 and 3 above, and we now leave the cluster whenever

$$\frac{p_{\text{Solved}}}{C_0 + C_\infty} \geq \frac{1 - p_{\text{Solved}}}{\gamma_\ell} \tag{7}$$

where ℓ is the last action inside the set $\mathcal{J}_0$ that is not yet performed.

If we leave the cluster some time before all the actions in $\mathcal{J}_0$ are performed, the observed cluster efficiency will be lower than the calculated one, since the members of the cluster were chosen to maximize this ratio. This means that we may have been too eager to enter the cluster, hence performing troubleshooting in a less than optimal way. To avoid this, we should consider the problem faced due to lack of information *before* deciding to enter the cluster. Therefore, we must find a set which is chosen to maximize the ratio according to (5), but with the additional requirement that neither (6) nor (7) should force us to leave the cluster before all actions in the set have been performed. We approximate this by a simple two-pass algorithm, which first finds the optimal set without considering lack of information by (5), then prunes this set. The result of this pruning is then used to approximate the cluster's efficiency.

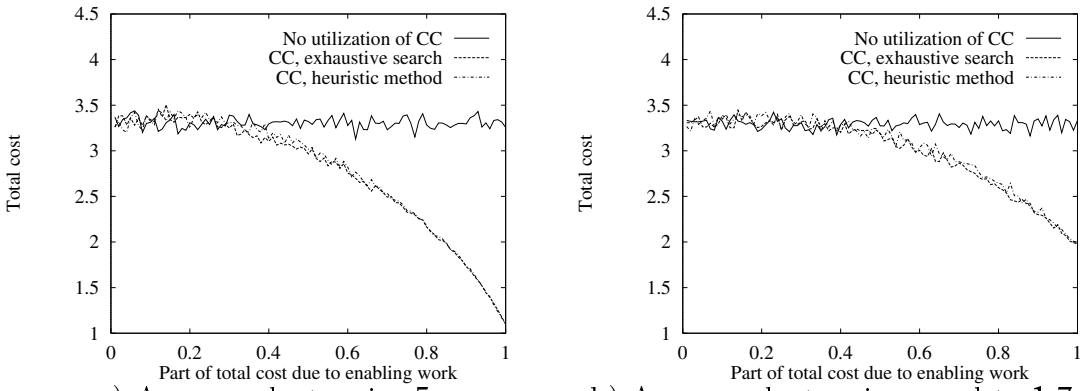
3.3 Empirical results

The formulas are implemented and tested on a small troubleshooter with $N = 10$ causes. Each cause has one dedicated action that remedies the cause with certainty. There is only one cluster with more than one action in the troubleshooter model; this cluster consists of from 5 to 9 of the actions, while the remaining actions are atomic. All actions have a total cost selected randomly with expected value 1. We varied the enabling cost so that it was from 0% to 100% of the total cost, leaving a reduced value for the cost of the problem-solving part of the action, γ_i.

The efficiency of the proposed heuristics was examined by comparing it to the results of normal troubleshooting and the optimal sequence found by an exhaustive search, see Figure 2. The average relative error (given as the difference of the heuristic ECR and the exhaustive search result divided by the result of the exhaustive search averaged over all data-points) is about 1.6% in the reported examples. Empirical results indicate that the error in the simulations scale fairly well, but the high computational complexity of the exhaustive search has prevented a thorough investigation at this point.

4 Dependent actions

By *dependent actions* we mean that the intersection of the parent sets of two actions is non-empty, i.e. they can both remedy one or more of the same faults. A simple example is given in Figure 1, where, e.g., both A_1 and A_2 can remedy the fault f_2. In this section we consider TS systems with dependent actions, but without conditional costs.



a) Average cluster size 5 b) Average cluster size equal to 1.7.

Figure 2: Average cost for the heuristic strategy (dash-dot) compared with the results of exhaustive search (dashed). The results of not utilizing conditional costs are also included (solid). Average cluster size are 5 and 1.7 respectively. The sample size is 1.000 samples per data point. The variance in the simulated estimates can be read of the graph by acknowledging the fact that the exhaustive search should *always* supply the superior results.

The standard approach would be to allocate to each action the total probability of it solving the problem at hand as indicated at the right hand side of Figure 1, and then to perform the actions one at a time always selecting the next action as the one with the highest efficiency given the updated knowledge. It is easy to see that this is not optimal in the case presented in Figure 1 if we assume all costs to be equal to 1. This greedy approach gives the sequence $\{A_2, A_1, A_3\}$ which has $\mathsf{ECR} = 1.5$, whereas the optimal sequence is $\{A_3, A_1\}$ with $\mathsf{ECR} = 1.45$. In this section we will propose a simple method for handling dependent actions. The idea is to utilize the *observation aspect* of the failed actions, and not only use the action's repair aspect to evaluate it.

Let the scaled efficiency ρ_i of action A_i be given by the action's efficiency divided by the sum of efficiencies, $\rho_i = \mathsf{ef}(A_i)/\sum_{j=1}^{N} \mathsf{ef}(A_j)$. S_i denotes the partial sequence after performing the first i actions, i.e. $S_i = \langle A_{i+1}, \ldots, A_N \rangle$. $S = S_0$ denotes the complete TSS. $\mathsf{ECR}(S_i \,|\, \boldsymbol{e}_i)$ is the ECR of a partial TSS $\langle A_{i+1}, \ldots, A_N \rangle$ with evidence that the actions $A_1, \ldots, A_i$ have failed. The value of information (VOI) captured in the event $\{A_i = n\}$ will be calculated as the decrease in $\mathsf{ECR}(S_i \,|\, \boldsymbol{e}_{i-1})$ when $\{A_i = n\}$ is appended to the knowledge base:

$$\mathsf{VOI}(A_i = n \,|\, \boldsymbol{e}_{i-1}) = \mathsf{ECR}(S_i \,|\, \boldsymbol{e}_{i-1}) - \mathsf{ECR}(S_i \,|\, \boldsymbol{e}_i)$$

This number is the gain in ECR obtained by *knowing* that A_i failed. The price reduction will be obtained if $A_i = n$, i.e. with a probability $1 - p_i$. Hence, we calculate the *"observation-biased cost"* of the action as $C_i - P(A_i = n \,|\, \boldsymbol{e}_{i-1})\ \mathsf{VOI}(A_i = n \,|\, \boldsymbol{e}_{i-1})$. (If $C_i - P(A_i = n \,|\, \boldsymbol{e}_{i-1}) \cdot \mathsf{VOI}(A_i = n \,|\, \boldsymbol{e}_{i-1}) \leq 0$, a small value $\delta > 0$ is used.) Employing this cost and not C_i rewards the observation aspect of the action. To include the repair aspect as well, we define the *observation-biased efficiency* as

$$\mathsf{obef}(A_i \,|\, \boldsymbol{e}_{i-1}) = \frac{P(A_i = y \,|\, \boldsymbol{e}_{i-1})}{C_i - P(A_i = n \,|\, \boldsymbol{e}_{i-1}) \cdot \mathsf{VOI}(A_i = n \,|\, \boldsymbol{e}_{i-1})} \tag{8}$$

The idea is now to use a greedy approach based on $\mathsf{obef}(\cdot|\boldsymbol{e})$ instead of $\mathsf{ef}(\cdot|\boldsymbol{e})$. This will change the efficiency allocated to the actions when we have dependent actions. If the requirements of Proposition 1 hold, however, the two efficiencies are identical.

Calculating $\mathsf{obef}(A_i \mid \boldsymbol{e}_{i-1})$ requires the calculation of $\mathsf{ECR}(S_i \mid \boldsymbol{\epsilon})$, which is time consuming. The rest of this section is devoted to approximating $\mathsf{obef}(A_i \mid \boldsymbol{e}_{i-1})$. An approximation of the optimal TSS is then found by a greedy search based on the observation-biased efficiency.

4.1 ECR *and Shannon entropy*

Recall the definition of Shannon entropy, $h(x_1, \ldots, x_N) = -\sum_{i=1}^{N} x_i \log x_i$, where $(x_1, \ldots, x_N)$ defines a probability distribution (see e.g. [7]). The entropy defines the level of "uniformness" within the probability distribution, as $h(\cdot)$ reaches its maximal value of $\log(N)$ for $x_i = 1/N$, $i = 1, \ldots, N$, and the minimal value of 0 when $x_i = 1$ for one i. It seems intuitive that the ECR of a TS-sequence is in part a function of the entropy of the scaled efficiencies. If the entropy is 0, we know what action solves the problem, and the ECR will be given by the cost of the only possible action. If the entropy is large, we do not have information to separate between the actions. The following proposition, proven in [5], formalizes this relationship.

Proposition 3 *For a troubleshooting model with an optimal sequence* S*, and where the entropy of the actions' scaled efficiencies equals* $h(\cdot)$*, there exists a sharp lower bound for the optimal* ECR *that is a function of* $h(\cdot)$ *only. This bound is strictly increasing in* $h(\cdot)$.

Similar results have been used for independent actions without the single-fault assumption, see e.g. [8].

We use the lower-bound of Proposition 3 to make an approximation of the value of the information we can get from a failed action. This approximation is

$$\widehat{\mathsf{VOI}}(A_i = n \mid \boldsymbol{e}_{i-1}) = \mathsf{ECR}_{\mathrm{Low}}(S_i \mid h(\boldsymbol{\rho} \mid \boldsymbol{e}_{i-1})) - \mathsf{ECR}_{\mathrm{Low}}(S_i \mid h(\boldsymbol{\rho} \mid \boldsymbol{e}_i)) \qquad (9)$$

were $h(\boldsymbol{\rho} \mid \boldsymbol{\epsilon})$ denotes the Shannon entropy of the scaled efficiencies given a knowledge base $\boldsymbol{\epsilon}$ and $\mathsf{ECR}_{\mathrm{Low}}(S_i \mid h(\cdot))$ is the lower-bound of the ECR of S_i given the Shannon entropy $h(\cdot)$. A computational scheme for this approximation, where the lower bound is approximated by a linear function, is given in [5].

4.2 The myopic approach

A computationally simpler approach to the estimation of $\mathsf{VOI}(\cdot)$ is to use the *myopic* approach. To approximate $\mathsf{ECR}(S_i \mid \boldsymbol{\epsilon})$ we order the actions in S_i by decreasing efficiency given $\boldsymbol{\epsilon}$. The ECR is then approximated by

$$\widehat{\mathsf{ECR}}(S_i \mid \boldsymbol{\epsilon}) = \sum_{j=i+1}^{N} C_j \cdot P(\boldsymbol{e}_{j-1} \mid \boldsymbol{\epsilon}) \qquad (10)$$

$\mathsf{VOI}(\cdot)$ is approximated by using (10) to approximate $\mathsf{ECR}(S_i \mid \boldsymbol{e}_{i-1})$ and $\mathsf{ECR}(S_i \mid \boldsymbol{e}_i)$.

4.3 Empirical results

In [5] a numerical example using a fairly large model (approximately 80 actions) shows that a greedy search based on the Shannon-approximation of the $\mathsf{obef}(\cdot)$ score reduces the ECR by more than 25% compared to that of a greedy search based on $\mathsf{ef}(\cdot)$. The myopic approximation of $\mathsf{obef}(\cdot)$ reduces the ECR by approximately 15%. Moreover, the greedy search based on Shannon-entropy was able to find the optimal TSS in Figure 1.

5 Conclusion

In this paper we have proposed heuristics to enable approximative algorithms to troubleshooting problems with conditional cost or dependent actions. The proposed methods are not exact, so the quality of the approximations are tested through some small numerical simulation studies. The conditional cost heuristic offers results that do not degrade the performance considerably as compared to the optimal solution found by exhaustive search. There are two heuristics for dependent actions; a myopic approach and one based one Shannon entropy. Both methods outperform the greedy search in an empirical study.

Acknowledgments

This work was supported by the Danish National Centre for IT Research through grant #87.2. We would like to thank our project coworkers, in particular Olav Bangsø and Marta Vomlelová-Sochorová, for interesting discussions.

References

[1] David Heckerman, John S. Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–56, March 1995. Special issue on real-world applications on Bayesian networks.

[2] Finn V. Jensen. *An introduction to Bayesian Networks.* Taylor and Francis, London, United Kingdom, 1996.

[3] Finn V. Jensen, Claus Skaanning, and Uffe Kjærulff. The SACSO system for troubleshooting of printing systems. In these proceedings.

[4] Jayant Kalagnanam and Max Henrion. A comparison of decision analysis and expert rules for sequential analysis. In P. Besnard and S. Hanks, editors, *Uncertainty in Artificial Intelligence 4*, pages 271–281. North-Holland, New York, 1990.

[5] Helge Langseth. Dependent actions: An approximation based on Shannon entropy, 2000. SACSO internal note.

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

[7] Joe Whittaker. *Graphical models in applied multivariate statistics.* Wiley, Chichester, 1990.

[8] Wang Xiaozhong and Roger M. Cooke. Optimal inspection sequence in fault diagnosis. *Reliability Engineering and System Safety*, 37:207–210, 1992.