POLYNOMIAL-TIME ALGORITHM FOR BASIC TROUBLESHOOTING WITH CALL-SERVICE

Václav Lín

Key words:
Decision-theoretic troubleshooting, polynomial-time algorithm, call service.

Abstract
In decision-theoretic troubleshooting, we are given a probabilistic model of a man-made device. Our task is to identify and eliminate a fault causing the device to malfunction through a sequence of troubleshooting actions. We study a variant of the problem where we can at any time "call service" and eliminate the fault at once by paying a fixed penalty. We give an $O(n \log n)$-time algorithm for the problem. This result is somewhat surprising, because it refutes an earlier conjecture that the problem has higher computational complexity.

Introduction

Assume that your computer crashes during boot. There is a number of things that you may do:

- Try to boot again several times.
- Try to boot in a "safe mode" of the operating system.
- Try to boot a different operating system from a back-up CD.
- Call a computer service.
- Etc.

Obviously, the options differ in their cost in terms of time or money, and in the probability of fixing the problem. It might a good idea to try to boot several times before we attempt to disassemble the computer or call a service man. In **decision-theoretic troubleshooting**, we solve combinatorial problems exactly of this kind. There exist various variants of the problem, see [2] for an introduction and [4] for thorough treatment. In this paper, we shall work on a mild generalization of the following simple scenario, called **basic troubleshooting**:

- There are n possible faults that can cause the device to malfunction: $F_1, \ldots, F_n$.
- At any moment of time, at most one fault can be present in the device (this is the so called **single fault** assumption).
- Each fault F_i, has a predetermined probability p_i of happening.
- There are n available repair actions $A_1, \ldots, A_n$, each bearing a cost c_i.
- Each action A_i solves fault F_i with certainty and does not solve any other fault.

Given information that the device modelled by this simple model is malfunctioning, our task is to devise a sequence of the repair actions with minimal expected cost. Each action in the sequence can succeed (i.e. fix the fault), or fail. When we perform an action that succeeds, we can stop the troubleshooting due to the single fault assumption. It is known [1,2] that the optimal sequence is obtained by selecting an yet-unused action A_i with the highest ratio p_i/c_i at each step of the sequence.

Now, assume that at each step of the troubleshooting sequence, we also have the option to give up and call a serviceman who is guaranteed to fix the device, albeit at a high cost. The authors of [2] conjectured that finding an optimal solution for this variant of the troubleshooting problem has a higher computational complexity than just sorting the actions, and left the problem open. In the next paragraph, we generalize the problem slightly, express it in formal terms, and solve it by giving an efficient algorithm. Thus we disprove the conjecture.

1. Problem statement and result

Assume a troubleshooting model with a set of **actions** $\mathcal{A} = \{A_1, \dots, A_n\}$, and a variable $\mathcal{F}$ with a set of mutually exclusive states $\{F_1, \dots, F_m\}$, $m \geq n$. The states of $\mathcal{F}$ are called **faults**. We take the **single fault** assumption – at most one fault can occur at any moment of time. Each fault has a nonzero probability $f_i \stackrel{\text{def}}{=} P(\mathcal{F} = F_i)$, $\sum_i f_i = 1$.

Each action can either fix the device when performed ($A_i = 1$), or fail ($A_i = 0$). Each action addresses exactly one fault and is not necessarily perfect (i.e. may fail to fix the fault):

$$P(A_i = 1 | \mathcal{F} = F_i) = q_i, \qquad q_i \leq 1,$$
$$P(A_i = 1 | \mathcal{F} \neq F_i) = 0.$$

We define $p_i \stackrel{\text{def}}{=} P(A_i = 1)$ and observe

$$p_i = \sum_{F_j} P\big(A_i = 1 \big| \mathcal{F} = F_j\big)\, P(\mathcal{F} = F_j) = q_i f_i.$$

Note that since $m \geq n$, there might be faults that are not addressed by any action. Since the actions are not perfect, $\sum p_i \leq 1$ even when $m = n$. That is, performing all the actions may not suffice to fix the fault.[1] Each action A_i has a fixed **cost** c_i.

Apart from $\mathcal{A}$, the model contains a designated action CS (**call service**) that always fixes the problem at cost c_{CS}.

Let $A_{\pi(1)}, \dots, A_{\pi(n)}$ be a permutation of actions. We use the following notation to express that the first k actions fail:

$$\epsilon_k \stackrel{\text{def}}{=} \big(A_{\pi(1)} = 0 \ \wedge\ A_{\pi(2)} = 0 \wedge \dots \wedge A_{\pi(k)} = 0\big),$$

Statement ϵ_k depends on the permutation π, but the permutation will be always known from the context. Since $P(A_i = 1 \ \wedge\ A_j = 1) = 0$ for $i \neq j$, we have

$$P(\epsilon_k) = 1 - \sum_{i \leq k} p_{\pi(i)}\,.$$

We also define an "empty statement" ϵ_0, with $P(\epsilon_0) = 1$. For the ease of notation, we write conditional probabilities as

$$p_{k|\epsilon_{k-1}} \stackrel{\text{def}}{=} P(A_k = 1 | \epsilon_{k-1}),$$

and $p_{k|\epsilon_0} \stackrel{\text{def}}{=} p_k$.

Note that $P(\epsilon_{k-1} | A_k = 1) = 1$ and therefore

$$p_{k|\epsilon_{k-1}} = \frac{P(\epsilon_{k-1} | A_k = 1) \cdot P(A_k = 1)}{P(\epsilon_{k-1})} = \frac{p_k}{P(\epsilon_{k-1})}.$$

Let $A_1, \dots, A_k$ be a permutation of a subset of $\mathcal{A}$. When we perform the actions in sequence, then, at each step, we perform the current action A_j, pay its cost c_j, and if the action fixes the device, we stop. If the action fails, we have to continue to the next action A_{j+1} – that happens with probability $P\big(A_j = 0 \big| \epsilon_{j-1}\big)$, and so on. This leads to the following definition of expected cost. The **expected cost** of performing the sequence $[A_1, \dots, A_k]$ is defined recursively as $ECR(A_1, \dots, A_k) \stackrel{\text{def}}{=} ECR(A_1, \dots, A_k | \epsilon_0)$, where

$$ECR\big(A_j, A_{j+1}, \dots, A_k \big| \epsilon_{j-1}\big) \stackrel{\text{def}}{=} c_j + P\big(A_j = 0 \big| \epsilon_{j-1}\big) ECR\big(A_{j+1}, \dots, A_k \big| \epsilon_j\big), \qquad j \geq 1$$
$$ECR(\emptyset | \epsilon_k) \stackrel{\text{def}}{=} 0.$$

[1] That is a generalization of the scenario described in the introduction.

If the sequence $[A_1, \dots, A_k]$ is followed by the call-service action CS, we have $ECR(A_1, \dots, A_k, CS) = ECR(A_1, \dots, A_k) + P(\epsilon_k)c_{CS}$. Note that $ECR(CS) = c_{CS}$.

REMARK. It is usual to define $ECR(A_1, \dots, A_k) \stackrel{\text{def}}{=} \sum_i P(\epsilon_{i-1}) \cdot c_i$, i.e. the cost of each action is multiplied by the probability of reaching it. It is easy to verify that this definition is equivalent to the recursive one.

DEFINITION**. Optimal troubleshooting sequence** is a sequence

$$[s^*, CS] = \arg \min_{\mathcal{B} \subseteq \mathcal{A},\ s \in \pi(\mathcal{B})} ECR\,(s, CS)$$

where $\pi(\mathcal{B})$ is the set of all permutations of $\mathcal{B}$.

REMARK. Note that:

- $\mathcal{B}$ can be an empty set,
- the probability of performing CS may be zero.

THEOREM 1**.** Optimal troubleshooting sequence can be obtained in $O(n \log n)$ time by the following algorithm:

Algorithm 1.

Define function **improveSequence**, which takes as input an action sequence $B_1, \dots, B_k$ (where possibly $B_k = CS$), and replaces the tail of the sequence by CS if it improves the ECR.

```
function improveSequence(B_1, ..., B_k):
    for j = k, k − 1, ..., 1 do:
        if c_CS < ECR(B_j, ... | ε_{j−1}) then:
            return [B_1, ..., B_{j−1}, CS];
    return B_1, ..., B_k;
```

Operation of the algorithm:

1. Order the actions in $\mathcal{A}$ so that the sequence of ratios $p_{\pi(i)}/c_{\pi(i)}$ is non-decreasing. Let $\mathbf{s} = [A_1, \dots, A_n]$ be the resulting sequence.
2. **do**:
   $\mathbf{s}' \coloneqq \mathbf{s}$
   $\mathbf{s} \coloneqq$ **improveSequence**$(\mathbf{s}')$
   while $(\mathbf{s} \neq \mathbf{s}')$
3. **return s**

REMARK. Our Algorithm 1 is an improved version of algorithm already proposed in [1]. However, paper [1] contains no proof of optimality of the algorithm.

2. Proof of the Theorem

LEMMA 1**.** Let $[A_1, \dots, A_k, CS]$, $k \geq 2$, be an optimal troubleshooting sequence. Then any pair of actions $A_j, A_{j+1} \in \mathcal{A}$ adjacent in the sequence satisfies $p_j/c_j \geq p_{j+1}/c_{j+1}$.[2]

Proof: Since $[A_1, \dots, A_k, CS]$ is optimal, it must be true that

$$ECR(A_j, A_{j+1}, \dots | \epsilon_{j-1}) \leq ECR(A_{j+1}, A_j, \dots | \epsilon_{j-1}),$$

$$c_j + P(A_j = 0 | \epsilon_{j-1})[c_{j+1} + P(A_{j+1} = 0 | (A_j = 0) \wedge \epsilon_{j-1})ECR(A_{j+2}, \dots | \epsilon_{j+1})]$$
$$\leq c_{j+1} + P(A_{j+1} = 0 | \epsilon_{j-1})[c_j + P(A_j = 0 | (A_{j+1} = 0) \wedge \epsilon_{j-1})ECR(A_{j+2}, \dots | \epsilon_{j+1})],$$

[2] This is well known, see e.g. [2,4].

$$-p_{j|\epsilon_{j-1}} c_{j+1} \leq -p_{j+1|\epsilon_{j-1}} c_j,$$
$$\frac{p_j}{c_j} \geq \frac{p_{j+1}}{c_{j+1}}.$$

■

LEMMA 2**.** Let $[A_1, \dots, A_k, CS]$, $k \geq 1$, be an optimal troubleshooting sequence. Then

$$\frac{c_k}{p_{k|\epsilon_{k-1}}} \leq c_{CS}\,.$$

Proof: Due to optimality of the sequence, $ECR(A_k, CS|\epsilon_{k-1}) = c_k + \left(1 - p_{k|\epsilon_{k-1}}\right)c_{CS} \leq c_{CS}$. That implies $c_k \leq p_{k|\epsilon_{k-1}} \cdot c_{CS}$.

■

LEMMA 3**.** Let $[A_1, \dots, A_k, CS]$, $k \geq 1$, be an optimal troubleshooting sequence. Then any action $A_x \notin \{A_1, \dots, A_k\}$ has $c_k/p_k < c_x/p_x$.

Proof: Assume to the contrary that $c_x/p_x \leq c_k/p_k$. With Lemma 2, that implies

$$\frac{c_x}{p_x} \leq \frac{c_{CS}}{P(\epsilon_{k-1})} < \frac{c_{CS}}{P(\epsilon_k)}\,,$$
$$\frac{c_x}{p_x} < \frac{c_{CS}}{P(\epsilon_k)},$$
$$c_x < p_{x|\epsilon_k} c_{CS},$$
$$c_x + \left(1 - p_{x|\epsilon_k}\right)c_{CS} < c_{CS},$$
$$ECR(A_1, \dots, A_k, A_x, CS) < ECR(A_1, \dots, A_k, CS).$$

That is a contradiction, since $[A_1, \dots, A_k, CS]$ is optimal.

■

Proof (of THEOREM 1): The algorithm returns a sequence of k actions, $0 \leq k \leq n$, followed possibly by CS. The returned sequence has two properties:

1. It contains exactly the k actions with highest efficiency ratio p_j/c_j, and
2. it is sorted by the efficiency ratio.

By Lemma 1 and Lemma 3, each of these two properties is a necessary condition of optimality. The algorithm ensures that:

- we consider all the $n + 1$ candidate sequences satisfying these necessary conditions,
- and the one with minimal ECR is returned.

Sorting the actions by p_j/c_j takes $O(n \log n)$ time [3], and the while-loop takes linear time.

■

Conlusions

We have solved an open problem proposed in the decision-theoretic troubleshooting literature. There are many variants of the troubleshooting problem that are known to be NP-hard (see [5]). In the future, we shall try to devise polynomial-time algorithms for special cases of these problems.

References:

[1] HECKERMAN, D., BREESE, J., ROMMELSE, K., *Decision-theoretic Troubleshooting*. CACM, 38:49-57, 1995.

[2] JENSEN, F. V., NIELSEN, T.D. *Bayesian Networks and Decision Graphs (2nd edition).* Springer-Verlag, 2007, 447 pages. ISBN-10: 0-387-68281-3.

[3] KUČERA, L., *Combinatorial Algorithms.* Bristol, England, Adam Hilger, 1989.

[4] OTTOSEN, T. J, *Solutions and Heuristics for Troubleshooting with Dependent Actions and Conditional Costs*, PhD thesis, Aalborg University, 2012. 198 p.

[5] VOMLELOVÁ, M. *Complexity of decision-theoretic troubleshooting*, International Journal of Intelligent Systems 18(2), pages 267-277, 2003.