Trouble Shooting

Lecture in Decision Support Systems #10

What is Trouble Shooting ?

Intro

Traditionally:

> Trouble shooting is finding (and offering advice with regard to correcting) the cause of a problem.

By implementing trouble shooting as a "pure" extension to Bayesian networks, it is possible to extend this functionality. It allows for a determination of the optimal sequence of steps to go through in the error-detection phase.

(Note that trouble shooting may be applied to more general problems, of the kind: What action should I take to maximize the gain.)

Trouble Shooting in Decision Trees

A decision tree is a classical way of making decisions. It consists of:

- A set of nodes, which may be a "choise-node" or a "probability-node".
- A set of edges. The edges going from "probability-nodes" contains weights (namely the probability of taking that decision), and the edges going from "choise-nodes" have costs assigned.

At the end of each branch is the sum of all the costs of the nodes, some of which have been weighted with the probability of the "probability-edges".

To find the optimal trouble-shooting sequence, one simply has to calculate all the sums at the leafs, and take the smallest. This task, however, may very easily be intractable, due to the exponential growth of decicion trees.

Trouble Shooting in Decision Trees

Example

In a situation with a device consisting of two components (of which one is observable), the decision tree looks in this way:

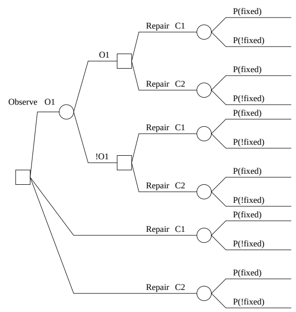

Figure 1: *Trouble shooting in a decision tree*

What is Trouble Shooting

Definitions

In a Bayesian network which models the problem, the following holds:

- The node representing the problem has a number of states, representing possible causes.
- for each state of the node, there is a repair action (a_i) connected to the problem node.
- Each repair action has a cost (c_i) associated with it.

What is Trouble Shooting

Example

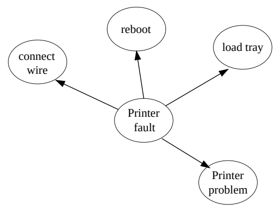

Figure 2: *A Bayesian network representing a problem in a printer.*

The "printer fault" node contains three states (Not connected, Driver error, Out of paper). The action nodes all contain two states (yes, no), and they represent the actions, which may repair each of the states in the "printer fault" node.

What is Trouble Shooting

Expected Cost

It is desirable to find the optimal sequence of repair actions. To measure the "value" of a sequence we use *Expected Cost*, referred to as EC.

For a sequence, S, of n repair actions, $EC(S)$ can be calculated in the following way:

$$EC(S) = \sum_{i=1}^{n} p(f_i) \sum_{j=1}^{i} c_j$$

Example:

	Not connected	Driver error	Out of paper
cost	1 min.	10 min.	2 min.
probability	0.5	0.4	0.1

For $S =< connectwire, reboot, loadtray >$

$$EC(S) = 0.5 * 1min + 0.4 * (1min + 10min) + 0.1 * (2min + 1min + 10min) = 6.2min$$

Expected Cost

Alternative

The expected cost may be calculated in another way.

For a sequence, S, of n repair actions, $EC(S)$ can be calculated in the following way:

$$EC(S) = \sum_{i=1}^{n} c_i * (1 - \sum_{j=1}^{i-1} p(f_j))$$

Example:
For $S =< connectwire, reboot, loadtray >$

$$EC(S) = 1min + (1 - 0.5) * 10min + (1 - (0.5 + 0.4)) * 2min = 6.2min$$

Exclusive Faults

If the following statements hold, there is an algorithm for calculating the optimal repair sequence.

1. For each state of the problem node, there is exactly one action node.
2. There is exactly one faulty node.
3. If the node f_i is faulty, the action a_i will repair it.
4. If the node f_i is faulty, the action $a_j, j \neq i$ will fail to repair it.
5. The probability of the node f_i being faulty is $p(f_i)$.
6. The cost of performing a_i is independent on the cost of performing a_{i-n}

The Greedy Algorithm

Description

The greedy algorithm calculates the sequence of repair actions, which yields the lowest expected cost.

This sequence is obtained by consistently choosing the next repair action to be the one with the highest probability/cost ratio:

1. calculate $\frac{p(f_i)}{c_i}$ for all a_i
2. perform the action with the highest score
3. if there are more actions to perform, goto 1

The Greedy Algorithm

Example

For the printer problem, with

	Not connected	Driver error	Out of paper
cost	1 min.	10 min.	2 min.
probability	0.5	0.4	0.1

The ratios are the following:

- Connect wire=0.5
- Reboot=0.04
- Load tray=0.05

This yields $S =< Connectwire, Loadtray, Reboot >$ and $EC(S) = 6$.

Exclusive Faults with Catchall

The greedy algorithm may work in cases, that does not comply with the requirements for exclusive faults. This is the case when the following holds:

1. For each state of the problem node, there is exactly one action node.
2. There is exactly one faulty node.
3. If the node f_i is faulty, the action a_i will repair it with probability p_i.
4. If the node f_i is faulty, the action $a_j, j \neq i$ will fail to repair it.
5. The probability of the node f_i being faulty is $p(f_i)$.
6. The cost of performing a_i is independent on the cost of performing a_{i-n}
7. There exists a repair action, a_s which will repair all faults, at cost c_s

Exclusive Faults with Catchall

Example

In the printer example, an extra action may be added, namely "call service", which will repair all the faults, with the cost 30 min. This leaves the following table:

	Not connected	Driver error	Out of paper	Call service
cost	1 min.	10 min.	2 min.	30 min.
probability	0.5	0.4	0.1	1

This leaves the following ratios:

- Connect wire=0.5
- Reboot=0.04
- Load tray=0.05
- Call service=0.03

This yields $S =< Connectwire, Loadtray, Reboot, CallService >$.

Trouble Shooting with Multiple Faults

When simultaneous faults may occur, a couple of modifications have to be made.

Let $\mathcal{E}$ be the evidence of the trouble shooter. It is initialized with the evidence, that the device is not functioning.

let p_i (the probability of a_i fixing the problem) be replaced by $p_i(\mathcal{E}_j)$ (the probability of a_i fixing the problem given the evidence $\mathcal{E}_j$).

However, this means that the repair probabilities are no longer constant, and thus EC can no longer be calculated !

Trouble Shooting with Multiple Faults

The Greedy Algorithm

The greedy algorithm now works like this:

1. let $n = 1$
2. calculate all ratios $p_i(\mathcal{E}_n)/c_i$
3. let $HighScore$ =the i of the highest scoring action
4. perform $a_{HighScore}$
5. let $\mathcal{E}_{n+1} = \mathcal{E}_n$ with the additional evidence, that $f_{HighScore}$ is fixed
6. let $n = n + 1$
7. if there are more actions to perform, goto 2

Trouble Shooting

Independent Faults

In some real life cases, the requirements of the exclusive faults may be to harse. In these cases, the greedy algorithm may be used as an approximation (as in the case of multiple faults).

However, if the faults are independent, a modification of the greedy algorithm will work.

Instead of choosing from the ratio $p(f_i)/c_i$, the ratio should be:

$$\frac{p(f_i)}{(1 - p(f_i)) * c_i}$$

and then the greedy algorithm will provide the optimal sequence.

Trouble Shooting

Observations

In many situations, it may be possible to optimize the trouble shooting sequence by making observations as it proceeds. We adapt to this situation by extending the Bayesian net with "observation nodes" , o_i. These nodes have a cost associated as well, oc_i.

In the greedy algorithm, we then choose to perform an observation, iff the value of the observation, $V(o_i)$, is larger than zero.

To calculate the value of an observation, the following steps are performed:

Let S_b be the optimal sequence without performing the observation.

let S_y be the optimal sequence when the observation yields "yes"

let S_n be the optimal sequence when the observation yields "no"

$$V(o_i) = EC(S_b) - (p(o = y)EC(S_y) + p(o = n)EC(S_n))$$

Trouble Shooting with Observations

Example

The printer problem is extended with an observation node, o_1, that observes the driver status (yes=loaded, no=not loaded). The cost of the observation is 1 minute.

	probability
o=yes	0.2
o=no	0.8

Table 1: The probability distribution for o_1

	Not connected	Driver error	Out of paper
cost	1 min.	10 min.	2 min.
$o_1 = y$	0.1	0.9	0
$o_1 = n$	0.6	0.1	0.3

Table 2: The cost/probability table for the printer problem, given o_1

The value of the observation can then be determined

Trouble Shooting with Observations

Example - continued

The optimal sequence if o_1 is not performed is $S_p =< Connectwire, Loadtray, Reboot >$ and $EC(S_p) = 6$.

If $o_1 = yes$, the sequence is $S_y =< Connectwire, Reboot, Loadtray >$
and $EC(S_y) = 0.1 * 1 + 0.9 * (1 + 10) = 10$

If $o_1 = no$, the sequence is $S_y =< Connectwire, Loadtray, Reboot >$
and $EC(S_y) = 0.6 * 1 + 0.3 * (1 + 2) + 0.1 * (1 + 2 + 10) = 2.8$

$$V(o_1) = 6 - (0.2 * 10 + 0.8 * 2.8) = 1.76$$

Thus we perform the observation

Trouble Shooting with Observations

The Myopic Approximation

When faced with the option of multiple choises and observations interleaved, the task of finding the optimal sequence may be computationally infeasable.

For this case, the *Myopic approximation* may be used.

When using the myopic approximation it is, at each step, assumed, that it is only possible to make an observation as the first step. In the future, no observations are allowed. Thus the greedy algorithm may be applied.

The Greedy Algorithm

Proof

Let $S = a^1, \ldots, a^n$, and S is ordered in decreasing order of the ratio p^i/c^i.

$$EC(S) = \sum_{i=1}^{n} p^i \sum_{j=1} ic^j$$

Let S' be S with the actions a^k, a^{k+1} swapped

$$EC(S') = \sum_{i=1}^{k-1} p^i \sum_{j=1}^{i} c^j + p^{k+1}(\sum_{j=1}^{k-1} + c^{k+1}) + p^k \sum_{j=1}^{k+1} + \sum_{i=k+2}^{n} p^i \sum_{j=1}^{i} c^j$$

This gives:

$$EC(S') - EC(S) = p^{k+1}c^k - p^k c^{k+1}$$

This means, that $EC(S') < EC(S)$ iff $\frac{p^k}{c^k} > \frac{p^{k+1}}{c^{k+1}}$