Decision-Theoretic Troubleshooting

David Heckerman John S. Breese Koos Rommelse

Introduction

You have just finished typing that big report into your word processor. It is formatted correctly and looks beautiful on the screen. You hit print, go to the printer, and nothing is there. You try again, still nothing. The report needs to go out today. What do you do?

Increasingly, computer users in this situation are calling customer support. Helpdesk services cost corporations hundreds of millions of dollars per year, and productivity losses due to users debugging software and hardware configuration problems probably are of a similar magnitude. As software becomes more complex and interoperable, the difficulty of diagnosis and repair grows. Other industries are experiencing similar increases in support and service costs for complex devices such as aircraft, trains, automobiles, and photocopiers.

Whereas the helpdesk may be able to solve your printing problem, you have had to wait for a response, communicate the problem to the technician, and ultimately track down the problem over a period of hours or sometimes days. Wouldn't it be better to have an automated local expert at your beck and call? This vision has been at the core of substantial research and development in the fields of artificial intelligence and expert systems. Since the early 1980s, diagnosis and treatment have been central problems in theoretical and applied AI [2]. Given that a device is not working properly or a patient has some complaint, the automated-diagnostic system is charged with determining the set of faults or diseases that explain the symptoms [3, 4, 6, 1]. The diagnostician is able to ask questions about the behavior of the device, or test individual components in order to determine if they are working properly. As new information is gained, the procedure updates its current view of the world. Inference focuses on identifying the set of faults consistent with the observations and ordering them in terms of likelihood. Information gathering proceeds until a single cause has been identified or the current diagnosis is sufficiently certain to support action.

Typically, however, our primary objective is to repair the device or cure the patient, not just determine what is wrong. At any stage of the process, there are many possible observations, tests, or repairs that can be applied. In addition, we may have the option of calling service: promoting the problem to a higher level of expertise that is guaranteed to be able to repair the device. Because these operations are expensive in terms of time and/or money, we wish to generate a sequence of actions that minimizes costs and results in a functioning device (or healthy patient). In this paper, we develop a diagnostic procedure that not only seeks to identify the most likely causes of a malfunction, but also generates a plan of action for repair. This plan consists of repairing or replacing individual components of a composite device or system, as well as making observations or tests. We and others call this process *troubleshooting* [3].

Optimal Troubleshooting and Decision Trees

An optimal troubleshooting plan is a sequence of observations and repairs that minimizes expected costs. The classic way to compute the expected cost of a plan is to use a decision tree.[1] In this section, we show how this computation is done. In the remainder of the paper, we introduce a more practical approach using Bayesian networks.

A decision tree represents the possible unfolding of events in temporal order. The representation contains two types of nodes: decision nodes and chance nodes. A decision node (drawn as a square node) represents a decision: an irrevocable allocation of resource. Branches of a decision node correspond to the mutually exclusive and collectively exhaustive set of alternatives available to the decision maker. A chance node (drawn as a circle) represents an uncertain variable. Branches of a chance node correspond to the mutually exclusive and collectively exhaustive possible states of the variable. Associated with each chance-node branch is the decision maker's probability that the variable will be in the corresponding state. Each path through the tree reflects a possible outcome for the decision maker. Associated with each path is the decision maker's preference for that outcome.

A decision tree for troubleshooting a simple two-component device is shown in Figure 1. The left-most square node represents the decision of whether or not to observe the variable o. The left-most circular node o represents an observation that provides some evidence about the status of the components. The number to the immediate left of a chance-node branch is the probability of that branch. The values at the end of the tree are the cumulative costs of observation and repair along the path from the root. For example, the second sum from the top of the figure $(5+10+20)$ is the cost of the path where first o is observed (cost $= 5$), then c_1 is repaired without fixing the device (cost $= 10$), and then c_2 is repaired (cost $= 20$). In constructing this model, we have assumed that the device is faulty initially, the device will be fixed if one repairs both components, and we can observe only o and whether or not the device is functioning. In addition, we have assumed that the cost of observing whether or not the device is functioning is zero, and all repair and observation costs are independent of the order in which actions are taken.

We compute the expected cost of a troubleshooting plan by *rolling back* a decision tree from right to left—a particular form of dynamic programming. At each step in the rollback procedure, we find a rightmost node, and compute the expected cost of the plan that would end at that node.

[1]By decision tree, we mean the representation described by Raiffa [9] for use in decision analysis.

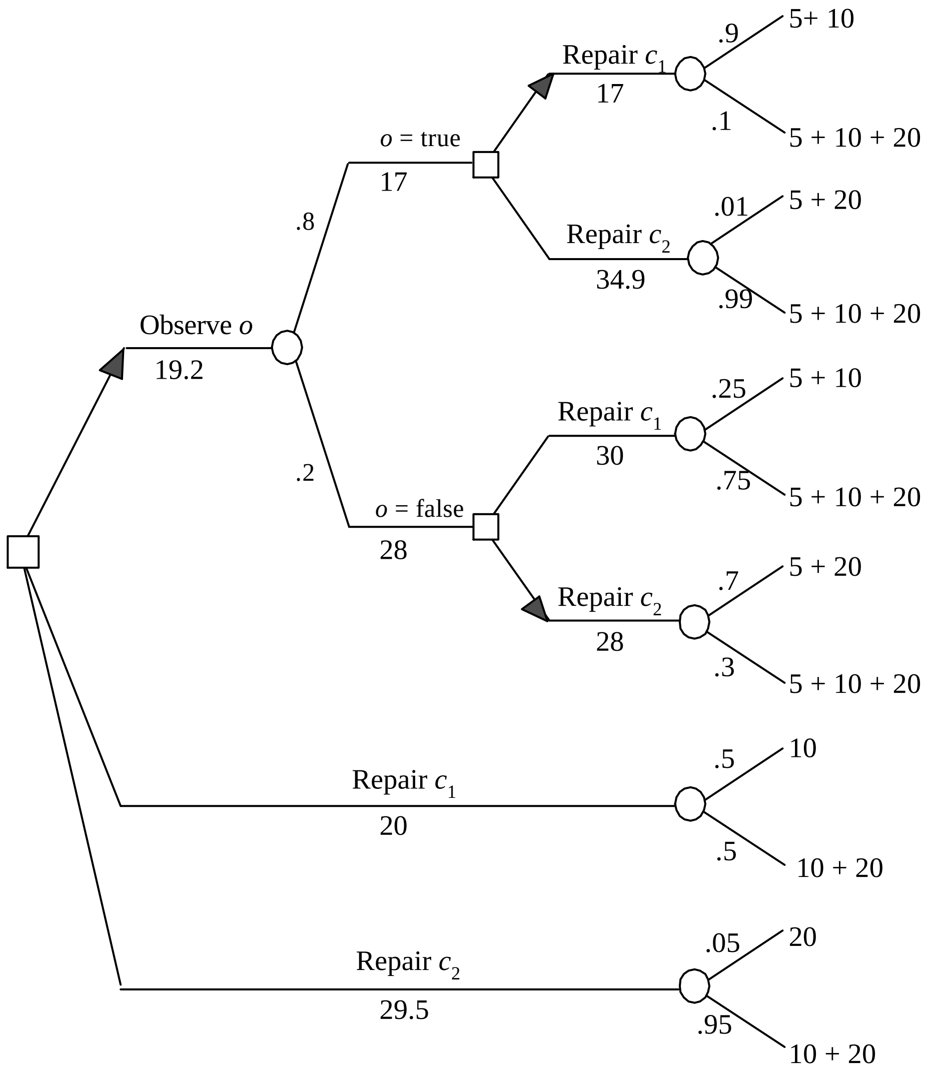

Figure 1: A decision tree showing all possible troubleshooting plans of a two-component device. The left-most square node represents the decision of whether to observe the variable o. The left-most circular node represents the variable o; and its branches represent the possible outcomes of the variables. As this is a chance node, we do not know which branch will be taken. The number to the immediate left of each of o's branches is the probability that the branch will occur. The rightmost column of chance nodes represent the observation of whether or not the device is functioning. The values at the end of the tree are the cumulative costs of observation and repair along the path from the root. For example, the second sum from the top of the figure $(5 + 10 + 20)$ is the cost of the path where first o is observed, then c_1 is repaired (not fixing the device), and then c_2 is repaired. To identify the plan with the lowest expected cost, we roll back the decision tree from left to right. At each step in the rollback procedure, we find a rightmost node, and compute the expected cost of the plan that would end at that node. The expected cost of each node is shown under the branch leading to the node.

When we encounter a rightmost chance node, we simply compute the expected cost of its branches. We do so by multiplying the number on each branch by its corresponding probability, and then summing the results for all branches. When we encounter a rightmost decision node, we find the branch of that node with the lowest expected cost, and set the expected value of the decision node to this cost. In addition, we record the action corresponding to the lowest-cost branch. The rollback procedure is illustrated in Figure 1. Under each branch is its expected cost as determined by the procedure. For example, the expected cost of the top branch labeled "Repair c_1" is

$$0.9(15) + 0.1(35) = 17$$

The expected cost of the branch labeled "$o =$ true" is

$$\min(17, 34.9) = 17$$

and the best action is "Repair c_1," indicated by the arrow. The optimal plan determined from a complete rollback says that we should first observe o; if $o =$ true, then we should repair c_1 first; and if $o =$ false, then we should repair c_2 first. This plan has an expected cost of repair of 19.2.

Development of an optimal solution to the general troubleshooting problem requires an analysis of all possible mixed observation–repair sequences using dynamic programming as shown above. As we increase the number of repairable components and possible observations, the decision tree grows exponentially. For example, a troubleshooting problem of this form with 5 components and 3 observations would generate a decision tree with nearly 340,000 endpoints. In this paper, we generate a series of approximations that more efficiently selects either an observation or repair action at each stage of the troubleshooting process.

Computing Probabilities of Faulty Components

When troubleshooting under uncertainty, we need to compute the probabilities that components have failed. In our example in the previous section, these probabilities are the numbers just to the right of the rightmost chance nodes. In our approach, we compute these probabilities using a *Bayesian network* (see introductory article). Figure 2 shows an example Bayesian network for a problem that, unfortunately, almost all of us have experienced: a car that won't start. As mentioned in the introductory article, most Bayesian networks are constructed by drawing arcs from cause to effect. This network is no exception. For example, "Battery Age" affects "Battery Quality," which affects "Battery Power," which affects "Engine Turn Over," which affects "Engine Starts."

Given observations of some nodes (e.g., "Engine Starts" is false and "Lights" are on), we can use a Bayesian-network inference algorithm to compute the probability that any or all of the system components are faulty. The conditional independencies represented by the Bayesian network make this computation practical in this case, and for most real-world problems as well.

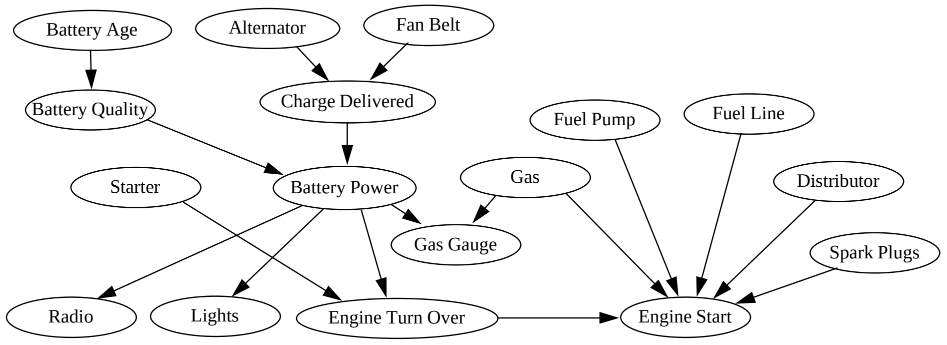

Figure 2: A Bayesian network for "my car won't start." Arcs are drawn from cause to effect.

If we repair a component and possibly make additional observations, we can still use this Bayesian network to compute the probabilities of component faults, but we must account for the change in the underlying state of the device and the fact that previous observations may have been invalidated. An efficient method for doing so is described by Heckerman et al. [5].

A Simple Special Case

In this section, we describe a set of assumptions under which it is possible to identify an optimal sequence of observations and repair actions in time proportional to the number of components in the device,[2] without explicitly constructing and rolling back a decision tree. We relax several of these assumptions in the following section.

The approach that we take is an extension of the troubleshooting approaches described in Kadane and Simon [7] and Kalagnanam and Henrion [8]. Let us suppose that our device has n components represented by uncertain variables $c_1, \ldots, c_n$, and that each component is in exactly one of a finite set of states.[3] In our automobile example, the components are "Battery Quality," "Alternator," "Fan Belt," "Starter," "Gas," "Fuel Pump," "Fuel Line," "Distributor," and "Spark Plugs." Our assumptions are as follows:

1. There is only one problem-defining node in the Bayesian network for the device. This node represents the functional status of the device. One of the states of this node must correspond to normal operation. In Figure 2, the node labeled "Engine Starts" is the problem-defining node.

[2]This time complexity assumes that Bayesian-network inference requires constant time. In practice, this assumption is often reasonable even though inference in an arbitrary Bayesian network is NP-hard (see sidebar).

[3]Our approach can be generalized to continuous variables, but we do not do so here.

2. At the onset of troubleshooting, the device is faulty. That is, the problem defining node is observed to be in a state other than "normal."

3. Single fault: Exactly one component is abnormal and is responsible for the failure of the device. We use p_i to denote the probability that component c_i is abnormal given our current state of information. These probabilities are computed using a Bayesian network, as described in the previous section. Under the single-fault assumption, we have $\sum_{i=1}^{n} p_i = 1$.

4. Immediately following any component repair, the problem-defining node is observed with cost C^p.

5. Each component is observable or unobservable. An observable component can be unambiguously tested or inspected to determine whether or not it is functioning properly. Furthermore, an observable component that is observed to be abnormal must be repaired immediately. An unobservable component can never be directly observed, but can be repaired or replaced. In our automobile example, the observable components are "Alternator," "Fan Belt," "Distributor," and "Spark Plugs." All other components are unobservable. For convenience, we use the phrase *observation–repair action* to refer both to the observation and possible repair of an observable component and to the repair of an unobservable component.

6. The costs of observation and repair of any component do not depend on previous repair or observation actions.

7. Limited observations: No other observations are available. In our automobile example, we do not permit the observation of "Battery Age," "Radio," "Lights," "Engine Turn Over," or "Gas Gauge."

For the moment, let us consider only observable components. Let C_i^o and C_i^r denote the cost of observation and repair of component c_i, respectively. If we observe and possibly repair components in the order $c_1, \ldots, c_n$, then for the expected cost of repair, denoted $\mathrm{ECR}(c_1, \ldots, c_n)$, we have

$$\begin{aligned}
\mathrm{ECR}(c_1, \ldots, c_n) &= (C_1^o + p_1(C_1^r + C^p)) + (1 - p_1)(C_2^o + \frac{p_2}{1-p_1}(C_2^r + C^p)) + \\
&\quad (1 - p_1 - p_2)(C_3^o + \frac{p_3}{1-p_1-p_2}(C_3^r + C^p)) + \cdots \\
&= \sum_{i=1}^{n} \left[\left(1 - \sum_{j=1}^{i-1} p_j \right) C_i^o + p_i(C_i^r + C^p) \right]
\end{aligned}$$

That is, we first observe component c_1 incurring cost C_1^o. With probability p_1, we find that the component is faulty and repair it (and the device) incurring cost $C_1^r + C^p$. With probability $1 - p_1$, we find that the component is functioning properly, and observe component c_2. With probability $p_2/(1 - p_1)$, we find that c_2 is faulty and repair it; and so on.

Now consider a troubleshooting sequence where we reverse the observation and possible repair of components c_k and c_{k+1}. All terms in the expected cost of repair of this sequence will be the same as those for the original sequence, except terms $i = k$ and $i = k + 1$. Therefore, we obtain

$$\mathrm{ECR}(c_1, \ldots, c_n) - \mathrm{ECR}(c_1, \ldots, c_{k-1}, c_{k+1}, c_k, \ldots, c_n) = p_{k+1} C_k^o - p_k C_{k+1}^o$$

Consequently, the sequence $c_1, \ldots, c_n$ has a lower (preferred) ECR than that with c_k and c_{k+1} reversed if and only if $p_k / C_k^o > p_{k+1} / C_{k+1}^o$. It follows that the optimal observation–repair sequence is given by the following plan:

1. Compute the probabilities of component faults given that the device is not functioning.

2. Observe the (as yet unobserved) component with the highest ratio p_i / C_i^o. Ties may be broken arbitrarily.

3. If the component is faulty, then replace it.

4. If a component was replaced, then terminate. Otherwise, go to step 2.

In this plan, if a component is found to be faulty and repaired, we know that the device must be repaired by Assumption 3. Consequently, we can terminate the troubleshooting process as specified in step 4. Also, note that fault probabilities need be computed only once.

Including unobservable components in our approach is straightforward. Recall that an unobservable component c_i is simply repaired with some cost R_i. Therefore, an unobservable component acts just like an observable component that is observed with cost R_i and always found to be faulty and repaired with cost zero. Consequently, we can include unobservable components in our procedure, provided we set C_i^o to R_i, and set C_i^r to zero.

Let us now examine some of our assumptions. Assumption 1 if often reasonable. When there is more than one "problem," we often can decompose our troubleshooting problems into two independent troubleshooting problems. For example, if our car does not start and our car side door is broken, then (for most cars) we can troubleshoot the problems independently. If there are interactions between faults, then the information that a second problem exists can be used in the probability calculations for troubleshooting the primary problem.

Assumption 2 usually is appropriate for troubleshooting, as there is no reason to use a troubleshooting system unless there is a problem with a device. In contrast, Assumption 2 may often be unreasonable in the context of automated systems for preventative maintenance.

Assumption 4 is often reasonable, except in those situations where the cost of testing a device is expensive. For example, when repairing a jet engine, it is often best to repair many components before retesting the engine.

Assumption 5 is almost always appropriate. Dividing components into those that are and are not practical to observe before repair comes with no loss of generalization. Furthermore, in the single-fault case, it is optimal to replace any component immediately after it has been observed to be faulty. In the multiple-fault case, this policy is not necessarily optimal, although it typically makes sense to repair a component immediately.

The validity of Assumption 6 depends on the problem domain. When trying to troubleshoot a printing problem, the costs of checking the driver software, network cable, power connection, and so on are reasonably independent of previous actions. When repairing an automobile engine, however, many components can be replaced with low cost once the engine header has been removed. We do not address methods for relaxing this assumption in this paper.

Approximations for More General Troubleshooting

In this section, we relax the single-fault assumption, and allow for more general observations in the troubleshooting plan. In addition, we consider the service-call action. To our knowledge, the generation of optimal troubleshooting plans allowing these extensions can not be done in time polynomial in the number of components. To handle these extensions, we introduce approximations based on our procedure described in the previous section. In Section , we describe experiments with real-world troubleshooting systems demonstrating that these approximations can lead to high-quality troubleshooting plans.

Service Call

Let us assume that, at any time in the troubleshooting process, we may call service. This action will have fixed cost C^s, and will lead to a functioning device with certainty. For example, a service call may be simply a replacement of the entire device. The assumption that the service cost is fixed is often reasonable in practice, although attempted repairs may decrease or increase service costs somewhat. Let us also assume that all components have repair costs less than the service cost. If this assumption is not true for a given component, then we simply replace a recommendation to repair the component with a recommendation to call service.

We can include the service-call action in our approach as follows. Because a service call is guaranteed to repair the device, it will always be the last action in a repair sequence. Furthermore, regardless of where in the sequence a service call occurs, the optimal observation–repair order for the remaining components is still determined by nonascending probability–to-cost ratios, as described in the previous section. Let us label the components so that the optimal sequence without a service call is $c_1, \ldots, c_n$. If we introduce a service call after the observation–repair of component k, we obtain

$$\mathrm{ECR}(c_1, \ldots, c_k) = \sum_{i=1}^{k} \left[\left(1 - \sum_{j=1}^{i-1} p_j \right) C_i^o + p_i (C_i^r + C^p) \right] + \left(\sum_{j=k+1}^{n} p_j \right) C^s \qquad (1)$$

To identify the position of the service call in the optimal sequence, we evaluate Equation 1 for each value of k, finding the value of $k = 0, \ldots, n$ for which the expected cost of repair is a minimum:

$$n_s = \min_k \left[\mathrm{ECR}(c_1, \ldots, c_k)\right] \tag{2}$$

where a service call is omitted from the plan if $n_s = n$. We can compute n_s in time linear in the number of components n.

We emphasize that plans generated by this approach are not necessarily optimal. In particular, we may be able to exchange the observation/repair of one or more components in $\{c_1, \ldots, c_{n_s}\}$ with that of components in $\{c_{n_s+1}, \ldots, c_n\}$ and obtain a plan with lower expected cost of repair.

Multiple Faults

The single-fault assumption is often a good approximation, because it is unlikely that two components will fail at roughly the same time. We see this behavior in our automobile example. Although the model permits multiple faults, when we observe that the car won't start, the failure of one component tends to explain away the failure of others. That is, the components are almost mutually exclusive when the device is faulty.

Nonetheless, there will be times when multiple components in a device have failed. In these cases, we can use the optimal single-fault plan with the following minor modification:

1. Compute the probabilities of component fault given the current state of information.
2. Observe the (as yet unobserved) component with the highest ratio p_i/C_i^o.
3. If the component is faulty, then replace it.
4. If a component was replaced, terminate if the device is working. Otherwise, go to step 1.

The only differences with the single-fault case are in steps 1 and 4. In step 4, we do not automatically terminate if a component was replaced. Rather, because there may be multiple faults, we terminate only if we observe that the device is working properly. Also, whether we observe the component to be working or repair the component without repairing the device, we go back to step 1 (rather than step 2), where we recompute the fault probabilities under our new state of information.

This plan is not necessarily optimal, because we incorrectly assume that there is only one fault in step 2, when we identify the next component to observe. Also, note that we can extend this plan to include service calls, using the procedure described in the previous section.

Nonbase Observations

So far, we have considered two special classes of observations: (1) the observation of the problem-defining variable after a repair is made, and (2) the observation of a component before a repair is made (as part of an observation-repair action). We refer to these observations as *base observations*. In many situations, we want to be able to make more general observations. For example, when our car fails to start, we may want to check the radio or the headlights in order to check the status of the electrical system. In this section, we describe a method for making such general observations.

Our approach is based on a second approximation. In particular, we pretend that we can make at most one nonbase observation before executing a plan consisting of only observation–repair actions and a service call. Then, we determine which nonbase observation if any should be made, and make the observation if appropriate. Finally, we iterate this procedure, possibly making additional observations. The procedure is sometimes said to be *myopic*, because we may make additional nonbase observations in the troubleshooting sequence, but we do not look ahead to these possible actions when selecting the next nonbase observation.

Suppose we have m nonbase observations $o_1, o_2, \ldots o_m$ available to us. Assume that observation o_i can take on exactly one of r_i possible states. We write $o_i = k$ to indicate that observation o_i takes on state k. In our myopic approximation, we first use the procedures described in Sections and to generate a troubleshooting sequence consisting of only base observations, repairs, and service call under the current state of information. Changing notation for simplicity, let ECR($\mathtt{I}$) denote the expected cost of repair of this sequence, where $\mathtt{I}$ is the current state of information. Second, imagine that we make some observation o_i first, and then determine the sequence of base observations, repairs, and service call. The expected cost of observing o_i with information $\mathtt{I}$, denoted ECO($\mathtt{I}, o_i$), is thus given by

$$\mathrm{ECO}(\mathtt{I}, o_i) = C_i^o + \sum_{k=1}^{r_i} \Pr(o_i = k|\mathtt{I})\ ECR(\mathtt{I} \cup \{o_i = k\}) \tag{3}$$

Note that the troubleshooting sequence following the observation may be different for every possible outcome of the observation. Finally, we repeat the computation of ECO for every possible nonbase observation.

If ECR($\mathtt{I}$) $<$ ECO($\mathtt{I}, o_i$) for every nonbase observation o_i, then we choose not to make a nonbase observation at this point in the troubleshooting process. Rather, we choose to perform an observation–repair action on some component or call service, as described in the previous sections. Otherwise, we choose to observe that variable o_i with the lowest ECO. After a repair or nonbase observation has been carried out, we update the information state $\mathtt{I}$, and repeat the cycle. A summary of this approach is given in Figure 3.

Empirical Results

We have applied our approach to troubleshooting printing problems, automobile startup problems, copier feeder systems, and gas turbines. The results have been satisfying along a number of dimensions. The models have been easy to build and assess. The generated plans in many cases

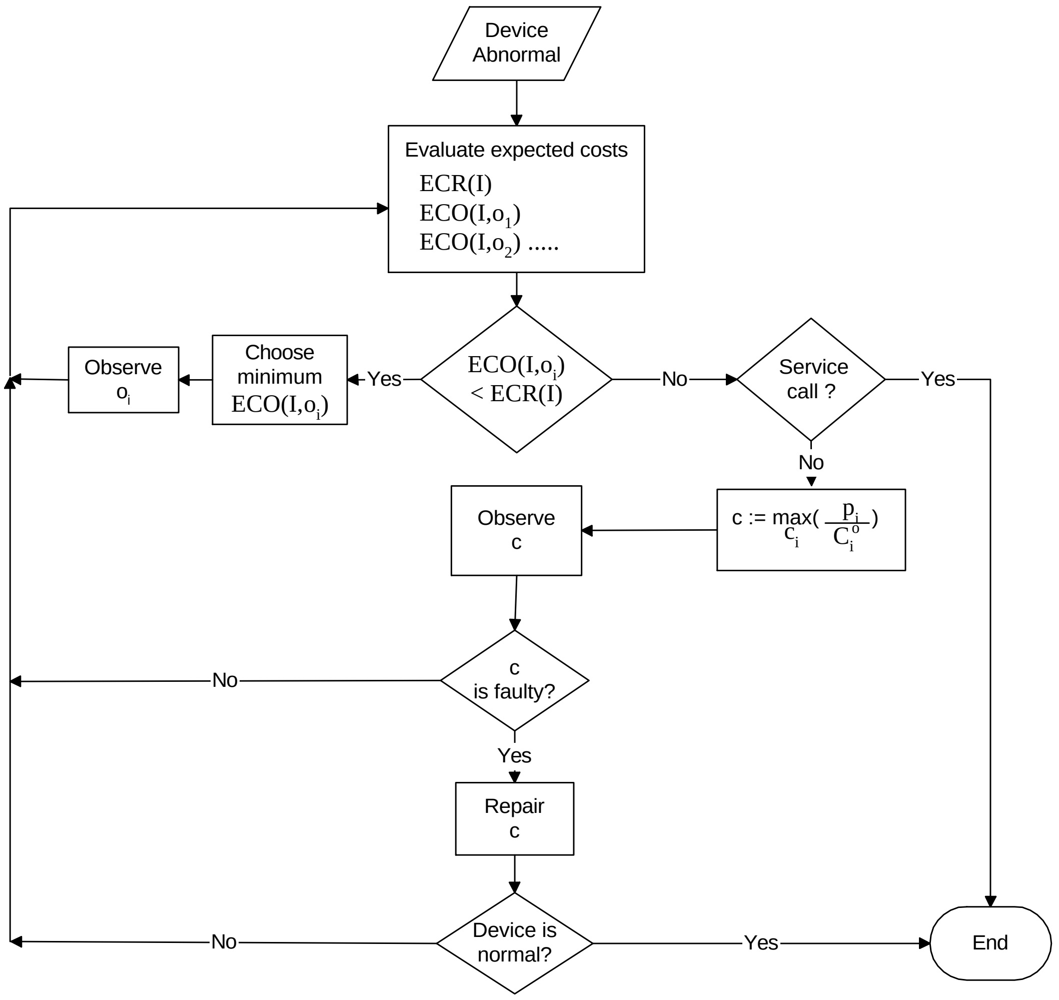

Figure 3: A summary of our approximate decision-theoretic method for generating a troubleshooting plan. First, we evaluate ECR($\mathtt{I}$)—the expected cost of repair under our current state of information $\mathtt{I}$—and for every observation o_i, the expected cost of its observation ECO($\mathtt{I}, o_i$). Next, if there is an observation o_i for which ECO($\mathtt{I}, o_i$) < ECR($\mathtt{I}$), then we observe o_i for which ECO($\mathtt{I}, o_i$) is a minimum. Otherwise, we check to see whether or not we should call service. We do so if and only if n_s, given by Equation 2, is equal to zero. If so, then we call service and then quit. Otherwise, we perform an observation–repair action on the component c_i such that p_i/C_i^o is a maximum. If we repair c_i, then we check if the device is functioning properly. If it is, then we quit, otherwise we repeat the process, identifying the next best action.

conform with intuition. In the remainder of this section, we discuss experiments that measure the performance of our decision-theoretic approach more precisely.

We have developed a Monte-Carlo technique for estimating troubleshooting costs for a given planner and domain. The basic idea is to use a Bayesian network for a given device to generate a relatively large set of problem instances where one or more faults are known to have occurred. We then apply the planning method to each case, recording the sum of costs of each action. A histogram of these total costs then provides a good estimate of the distribution of troubleshooting costs associated with a particular planner.

The method relies on an *oracle Bayesian network* to generate sample problems and to reveal the outcomes of observations given that specific components have been repaired. In our experiments, the joint probability distribution for the domain variables specified by the oracle Bayesian network is identical to that of the decision-theoretic planner, thereby insuring that the planner has the "correct" model. This assumption could be relaxed in future experiments. Our approach for generating cases guarantees that the problem-defining variable will assume an abnormal state in every case.

In the results that follow, we compare our decision-theoretic planner, a random planner, a static planner, and an omniscient planner in two domains: troubleshooting a car that won't start and troubleshooting the failure to print a document. Our decision-theoretic planner posts a mixture of repairs and nonbase queries to the oracle Bayesian network until the oracle reports that the device is repaired. The random planner posts repairs at random (without repetition) until the oracle reports that the device is repaired. The static planner posts repairs in a static order: components with lower observation costs are repaired first, with ties broken by repair cost. Again, the static planner continues until the oracle reports that the device is repaired. The omniscient planner knows exactly what faults are causing the device failure and repairs them. When posting repairs, each planner (except the omniscient planner) queries the oracle to first observe whether the device is faulty, and only repairs the component if it is defective. All planners can request a service call.

We generated 1000 troubleshooting cases for both domains. For the automobile problem, we used the Bayesian network shown in Figure 2 containing nine components and five nonbase observations. For the printing problem, we used the Bayesian network shown in the sidebar, containing 15 components and no nonbase observations. Figure 4 shows a histogram of the costs for both domains and for each of the planners for the 1000 cases.

In the automobile domain, the average cost of the omniscient, decision-theoretic, static, and random planners was \$127, \$154, \$298, and \$457, respectively. Thus, except for the omniscient planner, the decision-theoretic planner performed best, with the static planner coming in a distant third. The decision-theoretic planner saves \$144 per case on average over a static repair sequence. The total average cost for the omniscient planner sets a lower bound on the expected cost of an optimal planner. The decision-theoretic planner is close to this lower bound.

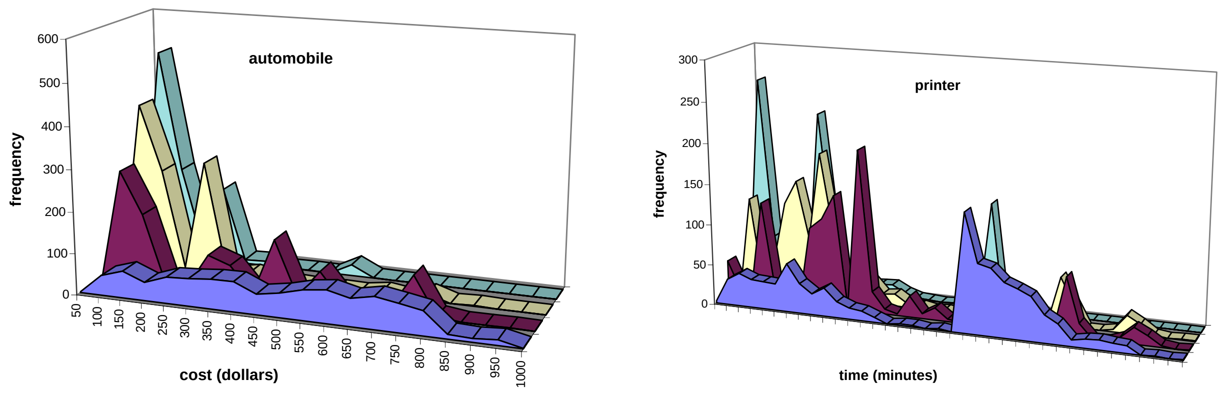

Figure 4: Cost histograms for the domains of automobile and print troubleshooting and for four planners. Shown from front to back are histograms for the random, static, decision-theoretic planners, and omniscient planners.

We obtained similar results in the printing domain. In this domain, cost was measured in minutes of delay. The average cost of the omniscient, decision-theoretic, static, and random planners was 32, 45, 56, and 84 minutes respectively. Again, except for the omniscient planner, the decision-theoretic planner performed best, with the static planner coming in third. Also, the decision-theoretic planner is relatively close to the lower bound set by the omniscient planner. For the decision-theoretic, static, and random planners, the secondary peaks in the histograms for the printer domain indicate plans where service was called. The omniscient planner rarely had to call service, as expected.

Thus, in both domains, the decision-theoretic planner had lower costs than either the static or random planner, and its repair costs were relatively close to the minimum possible repair costs. We note that in both domains, the variance of repair costs associated with the decision-theoretic planner were less than the variances associated with the heuristic planners.

In the automobile domain, the number of cases in which there was single, double, and triple faults was 930, 69, and 1, respectively. In the printing domain, the number of cases in which there was single, double, triple, and quadruple faults was 636, 285, 72, and 7, respectively. Therefore, in both domains, our single-fault assumption was quite good. Nonetheless, to investigate the effect of single versus multiple faults on our troubleshooting approach, we generated separate histograms for single- and multiple-fault cases in the automobile domain, shown in Figure 5. We see that the relative ordering of planners did not change. For the automobile domain, the savings for the decision-theoretic planner over the static ordering was \$144 over all cases. Among the single-fault scenarios, the average savings was \$138 whereas over the multiple fault scenarios the savings was \$208. Therefore, even though the single fault assumption was violated, the decision-theoretic

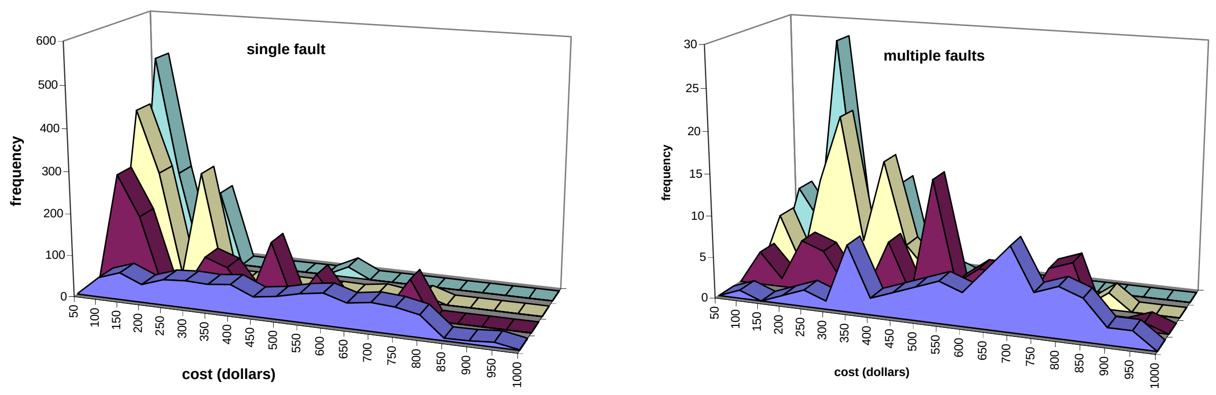

Figure 5: Cost histograms in the automobile domain for single- and multiple-fault cases. From front to back are histograms for the random, static, decision-theoretic, and omniscient planners.

planner did well, and, in fact, had a higher net savings because the average repair cost in multiple fault scenarios is higher. For the printer domain, we observed nearly the same average savings for both single fault and multiple fault scenarios.

Summary

We have described a decision-theoretic approach for generating troubleshooting plans under uncertainty that interleaves both observations and repair actions. Our approach is based on a set of approximations to an exact method for a simple special case. Despite our approximations, we have seen that our planner produces expected troubleshooting costs that are close to optimal and significantly lower than simple planners.

References

[1] J. Breese, E. Horvitz, M. Peot, R. Gay, and G. Quentin. Automated decision-analytic diagnosis of thermal performance in gas turbines. In *Proceedings of the International Gas Turbine and Aeroengine Congress and Exposition,* Cologne, Germany, American Society of Mechanical Engineers, June 1992.

[2] B.G. Buchanan and E.H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison–Wesley, Reading, MA, 1984.

[3] J. de Kleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[4] M. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:311–319, 1984.

[5] D. Heckerman, J. Breese, and K. Rommelse. Sequential troubleshooting under uncertainty. In *Proceedings of Fifth International Workshop on Principles of Diagnosis,* New Paltz, NY, pages 121–130, October 1994.

[6] D. Heckerman, E. Horvitz, and B. Nathwani. Toward normative expert systems: Part I. The Pathfinder project. *Methods of Information in Medicine*, 31:90–105, 1992.

[7] J. Kadane and H. Simon. Optimal strategies for a class of constrained sequential problems. *Annals of Statistics*, 5:237–255, 1977.

[8] J. Kalagnanam and M. Henrion. A comparison of decision analysis and expert rules for sequential diagnosis. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence,* Minneapolis, MN, pages 205–212. Association for Uncertainty in Artificial Intelligence, Mountain View, CA, August 1988. Also in Shachter, R., Levitt, T., Kanal, L., and Lemmer, J., editors, *Uncertainty in Artificial Intelligence 4,* pages 271–281. North-Holland, New York, 1990.

[9] H. Raiffa. *Decision Analysis: Introductory Lectures on Choice Under Uncertainty.* Addison–Wesley, Reading, MA, 1968.

About the Authors

DAVID HECKERMAN is a senior researcher in the Decision Theory Group at Microsoft Research. His research interests include the design of practical methods for constructing Bayesian networks from expert knowledge and learning Bayesian networks from data. **Author's Present Address:** Microsoft Research, One Microsoft Way 9S, Redmond, WA 98052-6399; email: heckerma@microsoft.com.

JOHN S. BREESE is a senior researcher in the Decision Theory Group at Microsoft Research. His research interests focus on developing tools and methods for decision-theoretic reasoning, in particular the integration of normative methods with symbolic processing techniques. **Author's Present Address:** Microsoft Research, One Microsoft Way 9S, Redmond, WA 98052-6399; email: breese@microsoft.com.

KOOS ROMMELSE is a researcher in the Decision Theory Group at Microsoft Research. His present interests are in the development of tools for probabilistic inference, troubleshooting, and learning Bayesian networks from data. **Author's Present Address:** Microsoft Research, One Microsoft Way 9S, Redmond, WA 98052-6399; email: koosr@microsoft.com.