Decision-Theoretic Case-Based Reasoning

John S. Breese
David Heckerman
(breese@microsoft.com,heckerma@microsoft.com)

November, 1994

Technical Report
MSR-TR-95-03

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

To appear in the *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics,* Fort Lauderdale, FL, January, 1995

Abstract

We describe a decision-theoretic methodology for case-based reasoning in diagnosis and troubleshooting applications. The system utilizes a special-structure Bayesian network to represent diagnostic cases, with nodes representing issues, causes, and symptoms. Dirichlet distributions are assessed at knowledge acquisition time to indicate the strength of relationships between variables. During a diagnosis session, a relevant subnetwork is extracted from a Bayesian-network database that describes a very large number of diagnostic interactions and cases. The constructed network is used to make recommendations regarding possible repairs and additional observations, based on an estimate of expected repair costs. As cases are resolved, observations of issues, causes, symptoms, and the success of repairs are recorded. New variables are added to the database, and the probabilities associated with variables already in the database are updated. In this way, the inferential behavior of system adjusts to the characteristics of the target population of users. We show how these elements work together in a cycle of troubleshooting tasks, and describe some results from a pilot system implementation and deployment.

1 Introduction

Most model-based [**?**] and probabilistic [**?**] methods for automated diagnosis use prespecified domain models as fundamental knowledge representations. The inference techniques used in these methods are powerful, but they rely on the availability of robust and relatively complete characterizations of potential problems, and have no explicit mechanism for feedback and learning from experience. Case-based reasoning (CBR) is a method for problem solving that uses a database of previously encountered problem solving incidents as its core representation. The presumption in CBR is that past problem solving behavior is the best predictor of future problems and solutions. In contrast to other diagnostic approaches, CBR typically utilizes less powerful reasoning methods, but is more responsive to the problem solving environment and requires less modeling effort. In this paper, we present a method that uses decision-theoretic and statistical techniques to provide normative inference and updating, while retaining the case-driven modeling that is characteristic of CBR.

Typically, CBR systems provide mechanisms to (1) index into the database of incidents to retrieve potentially relevant cases, (2) apply previous cases to the solution of the current problem instance, and (3) modify and update the case database to reflect new information and results from the current problem solving incident. In the literature, many techniques for representing and indexing cases, generating solutions, and updating the database have been developed [**?**, **?**].

In the approach described here, the database of previous cases is represented as a Bayesian network containing discrete variables [**?**]. The initial Bayesian network is constructed manually from an expert based on a set of past problem solving experiences. Cases are assumed to independent and identically distributed over some time horizon, and the uncertainty in parameters (i.e., long-run fractions) are encoded using Dirichlet distributions. At consultation time, input text describing the current problem is used to index into the relevant portion of the Bayesian network, corresponding to

Step 1 in the standard CBR process. The current implementation uses a probabilistic information-retrieval similarity metric to generate a list of potentially relevant symptoms and causes. Based on the user's selection, the relevant portion of the Bayesian network is constructed.

Probabilistic inference methods are then used to calculate the probabilities of various problems in the current case, and cost analysis is used to suggest possible repairs and information gathering actions. We refer to the result as a troubleshooting plan. In this portion of the consultation, which corresponds Step 2 in the standard CBR process, we use probabilities given by the means of the Dirichlet distributions stored in the Bayesian network.

Once a case is resolved, the Bayesian-network database is updated to reflect the new case. If the attributes observed in the case match those represented in the Bayesian-network database, then the relevant Dirichlet distributions in the network are updated to reflect the current observations. This allows the Bayesian-network parameters in the database to adjust to the characteristics of the target population. If attributes or observations in the current case do not match entities in the current Bayesian-network database, then the database is extended to include those variables with an initial set of human assessed parameters. These updating procedures correspond to Step 3 in CBR.

In the remainder of the paper, we describe the structure of the Bayesian-network database (Section 2). Then, we outline a cycle of decision-theoretic troubleshooting, including steps for database access, model construction, repair planning, and case recording (Section 3). Finally, we describe experiences with a pilot implementation of these methods in Section 4.

2 Bayesian-Network Structure

We have developed a stylized Bayesian network for case-based diagnostic applications. The basic organizing structure consists of nodes corresponding to causes, issues, and symptoms.

1. **Cause**: A contributing factor, activity, or configuration item—for example, "Gateway 4DX2-66" or "Printing Postscript File."

2. **Issue**: A conflict among a set of causes. Typically, the issue is defined to occur if all the associated causes occur—for example, "Gateway 4DX2-66V" and "BusLogic BT-445S SCSI/VL-Bus Adapter" and "Windows for Workgroups 3.11."

3. **Symptoms**: A particular behavior or malfunction caused by an issue—for example, "System will not boot," or "Cannot connect to network server."

The basic relationships among causes, issues, and symptoms in the Bayesian-network structure is shown in Figure 1(a) for a single issue. Issues may share common causes and symptoms as shown in Figure 1(b). For convenience, each issue is constrained to have at least one cause and at least one symptom.

Figure 1: (a) The Bayesian-network structure for a single issue, consisting of causes and symptoms. (b) The Bayesian-network structure for the case database, consisting of an interdependent set of issues.

Each cause is labeled as to whether it is fixable or unfixable. A fixable cause is one that can be altered in the course of troubleshooting. For example, in Figure 1(a), "Using MS/Style mouse port" is fixable with the action (or resolution) of "Switch to the PS/2 style mouse port." A cost is associated with each fix. In our formulation, an unfixable node corresponds to a fixable node with a very high fix cost. For example, the cause node "Windows NT 3.1" is treated as unfixable in the current model, though, in principle, one could switch operating systems at very high cost.

Symptoms and causes are labeled as being observable or unobservable. An observable node is one that the user can reasonably test or observe. For example, all the causes and symptoms in Figure 1(a) are observable. We assign a cost of observation to each observable in the database. Some causes—for example, defective hardware—may be unobservable yet fixable by replacing the component. In our formulation, symptoms are always observable.

All nodes in the Bayesian-network database correspond to discrete variables. The uncertainties associated with the parameters in the Bayesian-network database are encoded as Dirichlet distributions—in particular, means (probabilities for the next case) and equivalent sample sizes. The particular parameter means are as follows.

1. $\Pr(C_i)$ is the probability that cause i is present in a given case.
2. $Pr(S_k|I_j, \overline{I_i}, \forall i \neq j)$ is the probability of symptom k given only issue j is true.
3. $Pr(S_k|\overline{I_j})$ is the "leak" probability of symptom k given issue j is false.

In this formulation, we assume that there is a "noisy OR" relationship [**?**] between each symptom and the issues that cause it. This assumption makes both model assessment and inference more efficient [**?**, **?**]. Updating of the Dirichlet distributions is described in Section 3.5.

3 The Troubleshooting Cycle

The troubleshooting cycle is demand driven and consists of five basic steps:

1. Database Access: The text describing the problem is used to find potentially relevant symptoms or causes. One such variable is selected by the user to initiate the session.
2. Bayesian-Network Construction: The relevant portions of the Bayesian-network database are extracted for troubleshooting.

3. Bayesian-Network Solution: Using the constructed Bayesian network, the system generates recommendations for components to repair, and makes suggestions regarding additional observations.

4. User Execution: The user performs one or more repairs or observations, and reports the results to the system. The construction–solution–execution steps are repeated, until the problem is solved or the user suspends the session.

5. Case Recording: Upon termination, relevant Dirichlet distributions are updated. If variables not in the Bayesian-network database were observed, these can be added to the domain model, expanding the scope of issues covered.

In this section we describe each of these steps.

3.1 Database Access

A consultation is initiated by describing the problem with free text input. In the current test database, each cause, symptom and resolution is accompanied by a textual description. A full-text information-retrieval algorithm, similar to that proposed by Salton [**?**], is used to identify those causes or symptoms in the case database whose descriptions most closely match the textual problem description. The standard information-retrieval methods have been extended to deal with domain specific jargon and acronyms. Sets of synonymous technical terms are identified and the query is expanded. For example, in the test database, the terms "system error", "system trap", "blue screen", and "BSOD" are effectively merged in the search. Similar metods are used to deal with version numbers and the like [**?**]. The retrieval mechanism generates a list of symptoms and causes, ranked in order of similarity to the problem description.

3.2 Model Construction

As observations are made, the system identifies the portions of the Bayesian network that are relevant to the observations, constructing a Bayesian network that is a subset of the database. In this subset, we include all nodes that are dependent on the current set of observed nodes. In addition, because we are going to use the model to generate recommendations regarding questions to ask (i.e. perform value-of-information analysis), we also include portions of the network that would become relevant if we were to make observations.

Because we can observe any set of observable nodes in the model, the entire database is always potentially relevant to the current problem-solving task. However, as described in Section 3.3, we are using the constructed Bayesian network in a myopic value-of-information analysis that assumes that the diagnostician will make just one observation at a time. Therefore, we need to incorporate only those portions of the network made relevant by a single observation.

Figure 2: Three Bayesian-network fragments illustrating d-separation. (a) Observing a cause on a path between two subnetworks *blocks* that path. (b) Observing a symptom on a path between two subnetworks *activates* that path. (c) Observing a successor of an issue on a path between two subnetworks *activates* that path.

This construction rule is not sound, in principle, because the best observation to make may be completely unrelated to the current set of observations. This may occur, for example, if some problem is so likely to occur *a priori* that one should always inquire about that possibility, irrespective of the currently reported symptoms. Nonetheless, this situation occurs infrequently in many domains. Furthermore, the value of information of such variables is independent of the current consultation and may be computed off line.

We identify variables that are relevant to (i.e., dependent on) current observations using the graphical test of *d-separation*. In the following definition, a path refers to any undirected path.

Definition (d-separation, Pearl, 1988) *If* $\mathbf{X}$*,* $\mathbf{Y}$*, and* $\mathbf{Z}$ *are three disjoint subsets of nodes in a directed acyclic graph* $\mathcal{I}$*,* $\mathbf{Z}$ *is said to* **d-separate** $\mathbf{X}$ *from* $\mathbf{Y}$*, written* $< \mathbf{X}, \mathbf{Z}, \mathbf{Y} >_D$*, if there is no path between a node in* $\mathbf{X}$ *and a node in* $\mathbf{Y}$ *such that the following two conditions hold: (1) every node with converging arrows is in* $\mathbf{Z}$ *or has a descendant in* $\mathbf{Z}$ *and (2) every other node is outside* $\mathbf{Z}$*.*

A path that meets conditions (1) and (2) is said to be *active.* A path that does not meet these conditions is said to be *blocked.* We illustrate the use of d-separation as a graphical indicator of conditional independence in Figure 2. In Figure 2a, we have $< \mathbf{I}, \{C'\}, \mathbf{II} >_D$ but it is not the case that $< \mathbf{I}, \{\}, \mathbf{II} >_D$. That is, observing C' blocks all paths between the left and right subnetworks. In Figure 2b and c, we have $< \mathbf{I}, \{\}, \mathbf{II} >_D$ but not $< \mathbf{I}, \{S'\}, \mathbf{II} >_D$. That is, observing S' activates all paths between the left and right subnetworks. Thus, if two portions of the network are separated by a common cause, then observing that cause means that the probabilities of nodes in one subnetwork do not depend on observations in the other subnetwork and vice-versa. Conversely, observing a symptom that separates two subnetworks (or observing a symptom that is a successor of an issue that separates two subnetworks) means that the variables in the subnetworks are dependent on one another.

The model construction algorithm proceeds as follows.

1. From currently observed causes and symptoms, add all nodes and arcs on active paths from all observed nodes. This procedure stops exploring a path when either:

   (a) The path terminates at an unobserved symptom, or

Figure 3: (a) The implicit database network structure. The shaded nodes are the initial observations. (b) The constructed network after Step 1 has completed. Both the original and added nodes are shaded. (c) The final constructed network consists of the shaded nodes, where all ancestors of symptoms are added to the previous network.

(b) The path has a segment of the form $I \rightarrow S \leftarrow I'$ and S is unobserved, or

(c) The path has a segment of the form $C \rightarrow I \leftarrow C'$ and the successor symptoms of I are unobserved.

2. For each unobserved symptom in the constructed network, add all of its ancestors, if not already in the network.

3. End.

This process is illustrated in Figure 3 for an example Bayesian-network database. Note that additional portions of the database would become relevant only if we observed more than one additional symptom. Therefore, for purposes of myopic value-of-information, the currently constructed network is sufficient.

This algorithm can be viewed as a repeated application of an efficient method for finding sets of d-separated nodes developed by Geiger et al. (1990) . It also can be viewed as constructing a graph that is equivalent to the union of a set of networks that would be constructed by the knowledge-based model construction algorithm developed by Breese, where each symptom in the network constructed in Step 1 is provisionally observed [**?**]. Breese's algorithm forward chains from hypotheses (in this case the issues already in the network) and determines at each step if a node has been observed. If so, it adds all relevant upstream nodes, corresponding to Step 2 in the current algorithm.

3.3 Bayesian-Network Solution

The objective at this stage of problem solving is to generate a set of recommendations for repairs and observations based on the case at hand. In this paper, we adopt a scheme developed by Heckerman et al. (1994, 1995). In the following discussion, we first consider recommending just repair actions. We then address the possibility of observing additional symptoms and causes to focus troubleshooting.

Given a constructed network containing several possible fixable nodes, what order of repairs should be attempted to repair the device or system? In order to determine this sequence, we assume that there is a single *problem-defining* symptom, there is a single fault causing the problem defining node to be abnormal, repair costs are independent, and each fixable cause must be observed to

be abnormal immediately prior to its repair.[1] For our constructed network, one of the observed symptoms will be designated as the problem defining node.

Let C_i^o and C_i^r denote the cost of observation and repair of a fixable cause c_i, respectively. Let p_i be the probability that c_i is in a faulty state given current evidence, calculated using the Bayesian network constructed as described in Section 3.2. For purposes of generating a hypothetical repair sequence, as noted above, we make the assumption that there is a single fault and renormalize the probabilities of the fixable nodes from the Bayesian network such that $\sum p_i = 1$. If we observe and possibly repair components in the order $c_1, \ldots, c_n$, then the expected cost of repairing the system, denoted ECR is

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

That is, we first observe cause c_1 incurring cost C_1^o. With probability p_1, we find that the cause is faulty and repair it incurring cost C_1^r. With probability $1 - p_1$, we find that the cause is normal, and observe cause c_2 incurring cost C_2^o. With probability $p_2/(1-p_1)$, we find that c_2 is faulty and repair it; and so on.

Heckerman et al. (1995) show that the optimal troubleshooting sequence under these assumptions is given by observing and possibly repairing causes in descending order of the ratio p_i/C_i^o. Ties may be broken arbitrarily.[2]

This ordering assumes that there are no additional observations interleaved in the repair sequence. The additional observations, which we call *nonbase observations*, provide information about the system without changing the state of the system (i.e., without repairing causes). In order to estimate the value of these observations, we evaluate the expected cost of repair as if we can make at most one nonbase observation before executing a plan consisting of only observation–repair actions as described above.

This evaluation proceeds as follows. We assume that each observable 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. Let `E` denote the current information state of the troubleshooter. `E` may include information about previous observations as well as repairs. First, we use Equation 1 to compute the expected cost of repair for the plan where no nonbase observation is made, denoted ECR(`E`).

[1] These assumptions are discussed in detail in [?, ?]

[2] This procedure can be modified to include recommending a service call, i.e. some more sophisticated troubleshooting agent. See Heckerman et al. (1995) for more details.

Then, for one possible nonbase observation O_i, we determine the best observation–repair sequence for every possible outcome of O_i. For each possible outcome, we again use Equation 1 to compute the expected cost of repair of the best sequence, denoted $\mathrm{ECR}(\mathtt{E} \cup \{O_i = k\})$. Next, we compute $\mathrm{ECO}(\mathtt{E}, O_i)$, the overall expected cost of first observing O_i and then executing the observation–repair sequence:

$$\mathrm{ECO}(\mathtt{E}, O_i) = \\ C_i^o + \sum_{k=1}^{r_i} \Pr(O_i = k|\mathtt{E})\ \mathrm{ECR}(\mathtt{E} \cup \{O_i = k\}) \tag{2}$$

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

If $\mathrm{ECR}(\mathtt{E}) < \mathrm{ECO}(\mathtt{E}, O_i)$ for any nonbase observation O_i, then we choose not to make a nonbase observation and recommend the first action in the observation–repair sequence encoded in $\mathrm{ECR}(\mathtt{E})$. Otherwise, we recommend to observe that symptom or cause O_i with the lowest ECO. After a repair or nonbase observation has been carried out, we update the information state $\mathtt{E}$. If a repair action has been carried out, we must remove all observed symptoms that correspond to nodes that are descendants of the repair-action node, because the repair action may have caused these observations to change.

Although we may make additional nonbase observations in the actual troubleshooting sequence, we ignore these possible actions when selecting the next nonbase observation. Similarly, although the repair sequences are generated assuming a single fault, the iterative procedure allows us to deal with multiple fault scenarios. We have found this myopic approximation to yield good results in real-world problems [**?**].

3.4 Execution

Given the recommendations, the user performs one or more repairs or observations, and reports the results to the system. As additional variables are observed, it is necessary to reinvoke the model construction algorithm to extend the set of relevant causes, issues, and symptoms. Note that since repair actions may invalidate observations, the model could contract. Because our observation valuation procedure assumes a single additional observation at a time in the evaluation of Equation 2, the model construction algorithm is guaranteed to produce the correct probabilities. If we extended the myopic procedure to include k-step lookahead, then the model construction algorithm would need to similarly be extended.

3.5 Case Recording

Following a session, the database must be updated to reflect the observations in that particular case. At the end of a consultation, the Dirichlet distributions associated with observed nodes in

the database are updated. For simplicity, we only update a distribution if both its corresponding node and the parents of that node were observed. Thus, the "leak" probabilities for Noisy-OR distributions for the symptoms are not updated unless a symptom is observed to be true when no issue is confirmed.

In an operational system, we update the Dirichlet parameter distributions by pooling observations over a set of users. Consider a variable X with parents Y in the Bayesian-network database. Suppose that there are N_{jk} instances where $X = k$ and $Y = j$ in the current pool of observations. Then, assuming the long-run fraction of X given $Y = j$ has a Dirichlet distribution with equivalent sample size E_j, we update the mean and equivalent sample size of this distribution as follows:

$$p(X = k|Y = j) \longleftarrow \frac{E_j \cdot p(X = k|Y = j) + N_{jk}}{E_j + \sum_k N_{jk}}$$

$$E_j \longleftarrow E_j + \sum_k N_{jk}$$

In some cases, a new symptom, cause, or issue may be identified. In these cases, a corresponding node is added to the database, and arcs and Dirichlet distributions are assessed. If a new issue is identified, then several new causes and symptoms may also be added to the database.

In order to account for dynamics in the population of problems encountered, we can use a finite time horizon for Dirichlet updating—that is, discard older observations. A short horizon may be appropriate for new product introductions or revisions where we want to discount the effects of earlier observations, whereas a longer window may be appropriate for stable products and user populations.

4 Implementation

Two separate pilot projects codenamed "Aladdin," which use the techniques described in this paper, were undertaken at Microsoft during 1994 and 1995. The first pilot was developed to address setup and warranty problems for Microsoft Windows NT$^{\text{tm}}$ Version 3.1 operating system. The second pilot addressed support issues for various versions of Microsoft Word for the MacIntosh.

Table 1 shows information on the sizes and interconnectivity of the pilot datasets. The databases were constructed by Microsoft support engineers using a custom authoring tool. For these pilot projects, the authors translated existing textual problem description articles into the cause/issue/symptom framework used in decision-theoretic case-based reasoning. The Macword database has about 750 issues, taken from the most frequently accessed articles in the current information retrieval oriented product support tool.

One indicator of the utility of this form of knowledge representation is the degree of datasharing among multiple issues. For example, we see that for the MacWord database, on average a cause appeared in two separate issues. In both databases, the average in-degree of a symptom was close to one. This indicates that typically symptoms were entered in such a way that they were diagnostic

Node	*Windows NT*			*MacWord*		
Type	Number	In-Degree	Out-Degree	Number	In-Degree	Out-Degree
Causes	154	NA	1.25	927	NA	2.10
Issues	122	1.58	1.43	757	2.48	1.16
Symptoms	127	1.38	NA	901	1.02	NA

Table 1: Size and average in-degree and out-degree for causes, issues, symptoms in the pilot databases. NA = Not applicable.

of a particular issue. Over time, authors learned to enter general as well as specific symptoms, and use the inference mahinery to suggest more specific symptoms.

Some causes were very specific, but shared causes tended to be product version numbers and common activities, e.g. printing a file. The MacWord database has a higher cause out-degree and issue in-degree than the NT database because the MacWord issues typically contain a cause describing the product version while the NT database was for a single product version. Assuming that the desciption of all causes and symptoms are the same size, the Aladdin description of the issues is from 24 (NT) to 36 (MacWord) percent smaller than an identical flat description where cause and sysmptom descriptions are not shared.

The pilot advisory system was used by Microsoft support personnel for NT support issues for a two-month period in 1994. The implemented version used an approximation to full Bayesian-network inference if the networks became too large. Overall satisfaction with the style of interaction and the problem solving support tool by pilot participants was very high when compared to the current text-based information retrieval tools for diagnostic support. Unfortunately, the pilot knowledge base did not have sufficiently wide coverage for the range of issues actually encountered in on-line use, and therefore was not used extensively. Although Dirichlet parameter updating and model extension facilities were implemented and deployed, empirical results regarding effectiveness of these portions of the design are not yet available.

In order to investigate the utility of the techniques while controlling for coverage, we developed an experiment comparing the Aladdin for NT issues to the currently available information tools provided to support engineers. In these experiments, the correct solutions were available in both tools. The results in these test indicate that engineers using Aladdin could support products for which they had little or no training. In comparison to information retrieval, the probabilistic case-based reasoning approach was faster and resulted in fewer unsolved problems.

5 Summary

We have developed a design for a case-based reasoning system that uses a Bayesian network for runtime problem solving. The primary components of the system are an information retrieval mechanism to index into a Bayesian-network database, a data-driven model-construction algorithm, a

method for generating recommendations for observations and repairs from a Bayesian network, and a method for updating the Bayesian-network database as new cases are encountered. Preliminary results suggest that the method is useful as a troubleshooting aid.

Acknowledgments

The authors thank Eric Horvitz and Greg Shaw for useful comments and assistance on this work.

References

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

[2] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.