Automated Knowledge Elicitation and Flowchart Optimization for Problem Diagnosis

Alina Beygelzimer, Mark Brodie, Jonathan Lenchner, Irina Rish
IBM Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
{*beygel, mbrodie, lenchner, rish*}*@us.ibm.com*

Abstract

The established procedure for problem diagnosis in a wide variety of systems is often embodied in a flowchart or decision tree. These procedures are usually authored manually, which is extremely expensive and results in flowcharts that are difficult to maintain and often quite inefficient. A better diagnostic procedure would be one that automatically modifies itself in response to the frequency with which symptoms and underlying problems occur, in order to minimize the average cost of diagnosis.

We describe an approach to constructing a Bayesian network representation of diagnostic flowcharts, and demonstrate a system to support call center diagnostics based on this representation. One of the advantages of our approach is that it allows automated knowledge elicitation from "legacy" flowcharts. Using the new representation, knowledge is easier to author and maintain. By using information gain as a search heuristic, nearly-optimal flowcharts can be generated in response to data about the frequency of system faults or symptoms. The approach allows both prior expert knowledge and training data to be used to automatically generate and maintain flowcharts that respond flexibly to changing circumstances.

1 Introduction

The established procedure for diagnosing a problem in a faulty system is often embodied in a flowchart or decision tree. The system can be a piece of hardware, a piece of software, or a combination of hardware and software components. The diagnostic procedure may be executed by support personnel at a call center, by a voice response unit, by a web application when the system user seeks self-help, or it may even be executed automatically by the system itself in self-healing environments.

A flowchart is a natural way of representing the knowledge needed to diagnose problems. If the flowchart is comprehensive, it is usually easy for a human, even a non-expert, to follow the flowchart and diagnose the problem. At each node the human (or machine) can elicit the necessary information and decide which branch to follow, until a leaf is reached at which no more information is needed and the diagnosis is obtained. Flowcharts are a very good way of documenting the knowledge developed over time by people in resolving complex problems using their experience and expertise.

However, flowcharts suffer from a number of difficulties that restrict their utility for many applications. Firstly, they are quite difficult to author manually. It is quite expensive to obtain the necessary knowledge from human authors, because a large number of possible branches need to be considered to create a comprehensive flowchart. Even if a good initial flowchart can be manually built, maintaining the flowchart is an endless source of further difficulty. For example, every time a new type of fault is discovered, it needs to be added to the flowchart. A human being will at best typically add a new fault so that it does not take too long to diagnose and does not cause too much modification of the underlying flowchart structure. This can quickly result in the flowchart becoming unmanageably complex and incomprehensible.

A second major problem with manually authored flowcharts is that they are almost always sub-optimal. An optimal flowchart is one that minimizes the average cost of diagnosis, e.g., the average number of questions: common problems should be diagnosed more quickly by asking about them first, before asking about less common problems. To maintain optimality, a diagnostic flowchart must necessarily modify itself in response to changes in the frequency with which symptoms and problems occur. Manually authored flowcharts tend to become increasingly sub-optimal over time because of the difficulty of maintaining them. A particularly annoying example of this is when customer help personnel ask many unnecessary questions when trying to diagnose a problem.

It is of course possible to construct a flowchart using traditional decision-tree learning from training data. However, training data can be difficult to obtain (a working diagnostic system must already be in place before any training data can be collected). Furthermore, this approach fails to take advantage of the knowledge of human experts. People often have a very good understanding of what information should be elicited to perform the diagnosis, but they are usually unable to arrange the questions in the optimal order, given the frequency of the problems and symptoms and the complexity of considering all possible diagnostic paths.

In this work we describe an approach that automatically builds an alternative representation of the knowledge underlying diagnostic flowcharts. Namely, it creates a simple Bayesian network that is consistent with the available "legacy" flowcharts without asking an expert to go through the Bayesian net creation process. This representation has a number of advantages:

1. Knowledge is easier to maintain and update - authors simply need to specify the new questions or tests they might want to ask and, if available, what answers each test might yield, depending on the state of the system. No ordering information on the tests is needed, although ordering constraints can be provided if desired. Faults, tests and symptoms can be easily added, deleted and modified.

2. An efficient flowchart can be generated using a simple greedy algorithm that selects the order of tests, taking into account the frequency with which symptoms and problems occur. This flowchart can be shown to be close to the optimal flowchart obtained by exhaustive search.

3. The generated flowchart changes automatically as data about the frequency of faults and symptoms is obtained from use of the diagnostic system. Thus both prior human knowledge and training data are leveraged to allow for continuous learning of efficient diagnostic procedures.

4. Any pre-existing ("legacy") flowchart can be easily converted into the new representation. This allows us to take advantage of human expertise by creating an optimized version of any existing flowchart. Two types of optimization are particularly common: unnecessary tests are removed, and the order of questions may change as the problem frequency changes.

We also describe a system we have built to support call center diagnostics based on this new authoring paradigm. The system is called FLOAT, for Flowchart Learning, Optimization, Authoring and Testing. Authors simply describe their knowledge of states and symptoms (or they can convert an already existing flowchart). Once a flowchart or decision tree is generated, authors can test out the flowchart interactively and modify the knowledge if needed. We provide examples illustrating the use of the system to improve existing flowcharts.

2 Problem formulation

Given a set S of n possible states of a system, such as possible faults, we want to distinguish between those states by probing for symptoms that differentiate between the various possible states as efficiently as possible. We refer to such probes variously as questions or tests, and denote the set of all available probes by Q. Some $q \in Q$ may be more expensive to administer, or ask, than others. It is in fact more precise to speak of the cost of answering a question or responding to a probe, and it may in fact be that the cost of asking a question is dependent in part on the answer given. In what follows, we ignore this nuance and assume that the cost of answering a question is the same for all answers. Thus each question q has an associated cost $c(q)$.

Each test $q \in Q$ corresponds to a disjoint collection of subsets $U_1, ..., U_k \subset S$, with the interpretation that the question q can be answered in one of k possible ways $a_1, ..., a_k$, and given the answer a_i the actual state s of the system may either be contained in U_i - the collection of states for which we know the answer to be a_i, or in $S \setminus \bigcup U_i$ - those states for which we are not certain which of the answers to q applies.

We seek an efficient sequence of tests that is guaranteed to distinguish between every two elements of S. We can find such a sequence only if the questions in Q are capable of differentiating all elements of S. Q is then said to be **separating**. If Q is not separating, at the end of asking a sequence of questions, we will necessarily be left with a probability distribution over the remaining states.

3 Flowcharts

A **flowchart**, or decision tree, T for the set S of possible states is a tree with the elements of S as leaves, a separating subset of Q as internal nodes, and, given an internal node $q \in Q$, the edges below q are given by the possible answers a_i to q. With some abuse of notation we shall sometimes write that $q = a_i$ meaning that in the given context, the question q was given the answer a_i. It is possible that an element in S can appear as multiple leaves in a decision tree T, and equally that an element of Q can exist at multiple nodes.

A **path** p through T is given by a sequence of questions along with a final state: $p = \langle q_{i_1}, ..., q_{i_k}, s \rangle$. The cost of a path p through T is then given by $c(p) = \sum_{j=1}^{k} c(q_{i_j})$. Given a probability distribution on the states (or paths in the event that some states appear multiple times as leaves), the **cost** of the decision tree T is the expected cost of diagnosis, i.e. $c(T) = \sum_p \Pr(p)c(p)$.

It is also possible to consider trees which do not necessarily terminate in states, but rather, in some cases, in probability distributions over sets of remaining states. We call such a tree a **non-separating tree**. The set of all internal nodes of a non-separating tree may or may not be a separating set of questions. In what follows we assume we always have a separating set of questions, and we require that all proper decision trees T be separating.

The basic structure of T can be represented somewhat differently by collapsing into a single node any question or state that appears multiple times in the tree, yielding a directed acyclic graph or DAG. There is also the notion that one graph can call another graph, and in so doing even create cycles. In this most general conception, the underlying graph is called a flowchart. We shall take the point of view that cycles are never beneficial, and so all flowcharts can be represented as DAGs. Further, when a question can be asked via multiple paths, we replace the question with multiple instances, one per path, and similarly when a state can be reached via multiple paths. Thus, in all cases we equate a flowchart with a decision tree.

4 Alternative representations: Bayesian Networks and Dependency Matrices

The flowchart representation of expert knowledge in diagnostic applications seems to be quite traditional, but it suffers from several drawbacks, such as the complexity of modifying flowcharts when states, questions and their outcomes are added, deleted or changed, as well as the sub-optimality of the diagnosis process when using manually constructed decision trees. We propose below an alternative representation that eliminates the above drawbacks.

A Bayesian network approach to diagnosis, where the state corresponds to a combination of various faults in a system, was previously addressed in [7, 9], where a bipartite Bayesian network was used to model the dependencies between the component states and the probe outcomes, assuming that the probe outcomes are independent given the system state. This assumption actually holds in many diagnostic cases, and allows for very efficient knowledge representation and inference. In this paper, we use a simpler approach, treating the system state as one multi-valued variable, which effectively reduces the bipartite graph to a naive Bayes model (a state variable pointing to various probe variables).

Given the naive Bayes assumption, a Bayesian network can be represented explicitly by a so-called dependency matrix [7]. A **dependency matrix** D for a set S of n states and a set Q of m questions is an m x n matrix with rows corresponding to questions, and columns corresponding to states. The entry D_{ij} corresponds to the probability distribution of the answer(s) expected to question q_i, given that the system is actually in state s_j. If a question is not relevant to a state, the corresponding matrix entry is left blank (shown by an asterisk).

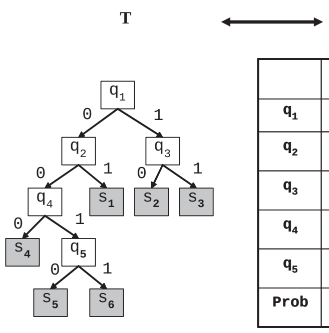
Dependency matrices can easily be modified incrementally, for example by simply adding a column in response to a new state, and as statistics are captured about the relative frequency of states or symptoms, the matrix values are automatically modified. A symptom is identified with an answer to a question.

Dependency matrices have been widely used for problem diagnosis (see [4], [6] and [7]). They are much more convenient for knowledge authoring and incremental modification, and moreover can be used to create almost optimal flowcharts or decision trees. They are also well adapted to learning from experience. As data accumulates about the prevalence of various symptoms or states, this information can easily be added to the dependency matrix.

Unlike a decision tree, a dependency matrix does not directly include information as to what question to ask given a set of remaining states, and possibly a probability distribution over those states. In order to exploit the advantages of the dependency matrix representation, we need to be able to convert dependency matrices into flowcharts.

5 Flowchart Optimization

First assume that all questions in Q have the same cost. We begin with a prior distribution over the set S of states; more generally at any point in the construction of the flowchart, we are at a node with a probability distribution over the set of remaining states $\widehat{S}$. A natural greedy algorithm, which we shall refer to as $GREEDY$, is to choose that $q \in Q$ which maximizes the expected information gain, or expected decrease in entropy, from answering q. If $H(\widehat{S}) = \sum_{s \in \widehat{S}}(-\Pr(s) \log \Pr(s))$ is the entropy associated with $\widehat{S}$, then the expected information gain by asking q, $I(\widehat{S}|q) = H(\widehat{S}) - \sum_{i=1}^{k} \Pr(q = a_i) H(\widehat{S}, q = a_i)$ where $\{a_i\}$ are the possible answers to q and $H(\widehat{S}, q = a_i) = \sum_{s \in \widehat{S}, q=a_i}(-\Pr(s) \log \Pr(s))$.



	s_1	s_2	s_3	s_4	s_5	s_6
q_1	0	1	1	0	0	0
q_2	1	*	*	0	0	0
q_3	*	0	1	*	*	*
q_4	*	*	*	0	1	1
q_5	*	*	*	*	0	1
Prob	.25	.25	.25	.125	.063	.063

Figure 1: Complementarity between decision trees and dependency matrices.

We select the $q \in Q$ which maximizes $I(\widehat{S}|q)$, add one edge for each of its possible answers a_i, create new nodes with updated probability distributions over the states, one for each answer a_i, and then repeat the process, selecting the most informative question at each of the new nodes. In this way all paths through the flowchart are constructed in parallel. A path ends either when only one state remains with non-zero probability or there are no remaining questions which have non-zero information gain. If questions have different costs, $GREEDY$ is easily modified to select the $q \in Q$ which maximizes $I(\widehat{S}|q)/c(q)$.

The flowchart optimization problem is NP-hard [2] and has no constant factor approximation algorithm [8]. The above $GREEDY$ algorithm was shown to be within a factor $O(\log n)$ of optimal in [5]. The authors showed that a slight re-weighting of leaf probabilities (provably needed if the probabilities are exponentially unbalanced) results in a tree whose cost is within a factor of $O(\log n)$ from optimal, for any distribution on the n leaves.

6 Knowledge Elicitation: Constructing Dependency Matrices from Flowcharts

Given a flowchart (decision tree) T, we now address the question of constructing a corresponding dependency matrix M that would incorporate the knowledge from T about possible system states and test outcomes. One natural approach would be to consider the flowchart as a set of constraints imposed on the joint distribution over the states and tests that will be represented by a corresponding dependency matrix M, and use a maximum-entropy approach to reconstruct M from T. Namely, every path from the root to a leaf in T implies a constraint on the tuples $(\mathbf{t}, s)$, where $\mathbf{t}$ is a set of all tests and s is the system state, requiring that the subset of tests along the path, as well as the system state variable s take the values specified by that path. One interpretation of the maximum-entropy approach would assume a uniform distribution over *all* $(\mathbf{t}, s)$ consistent with (at least) one such path constraint. Alternatively, we could assume a uniform distribution over *each* set of tuples corresponding to a particular path. Yet another approach is simply to assume, for every test not on the current path, a uniform distribution of test outcomes conditioned on a given state specified by the leaf of the path. While there are different arguments for and against each set of assumptions (which we hope to explore more in the future work), our current approach uses the last assumption and is briefly described below.

We assume the nodes of T to be labelled with their associated questions, and the edges to be labelled by the associated answers or symptoms. Since states are represented by leaves in T and columns in M, we populate the associated cells of M by traversing the path from the root of T to the associated leaf and reading off the answers to each question encountered. All other cells of the matrix are filled in with an asterisk indicating that the cell value is "unknown." The only trick is to assign a probability distribution to the states, and various approaches are possible as we mentioned above. For now, we assume at each stage that each answer to each question is equiprobable; then to obtain the probability of a state s we just multiply the probabilities of each answer along the path from root to s.

If we are able to assume that all leaves are associated with unique states, then $GREEDY$ applied to M will precisely yield T (or a flowchart completely equivalent to T) giving full "complementarity." A simple information-theoretic computation shows that $GREEDY(M) = T$ except in the case in which there are two questions q_1, q_2 which are completely interchangeable. q_1 and q_2 are interchangeable if first q_1 is asked, and regardless of the answer to q_1, q_2 is asked. Figure 1 illustrates the complementarity between the decision tree T and the matrix M. M is assumed to be equipped with a complete set of priors, as illustrated.

7 Implementation

We briefly describe a system we have created to support call-center diagnostics that utilizes the complementarity between decision trees and dependency matrices. The application is called FLOAT, for Flowchart Learning, Optimization, Authoring and Testing. The idea is that authors describe their knowledge of states and symptoms (i.e. answers to questions) using a dependency matrix. Alternatively, a dependency matrix can be created from an already existing flowchart. Then the $GREEDY$ algorithm described above is used to generate a flowchart from the dependency matrix, taking into account whatever information is available

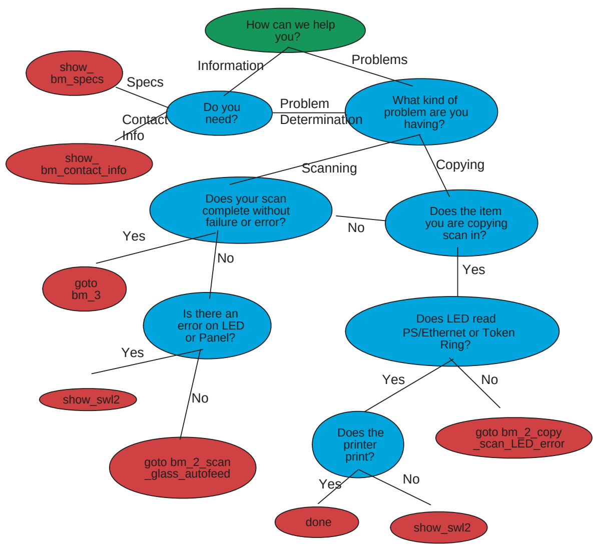
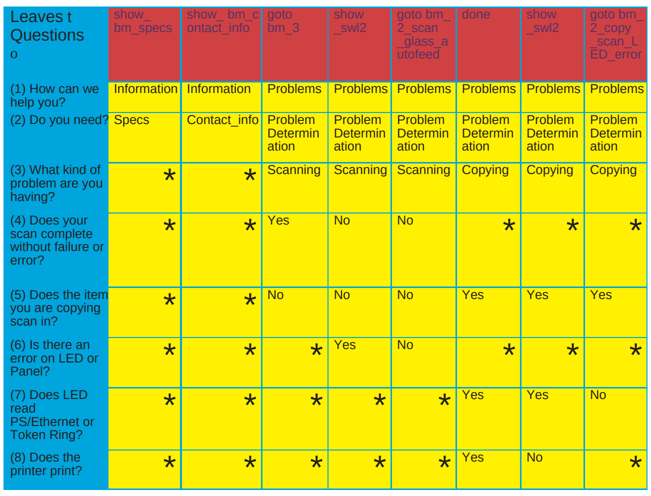

How can we help you?
Information
Problems
show_ bm_specs
Specs
Do you need?
Problem Determination
What kind of problem are you having?
Contact Info
show_ bm_contact_info
Scanning
Copying
Does your scan complete without failure or error?
No
Does the item you are copying scan in?
Yes
goto bm_3
No
Yes
Is there an error on LED or Panel?
Does LED read PS/Ethernet or Token Ring?
Yes
show_swl2
No
Yes
No
goto bm_2_copy _scan_LED_error
Does the printer print?
goto bm_2_scan _glass_autofeed
Yes
No
done
show_swl2

Figure 2: A piece of a real flowchart.

about the frequency of the different states. Authors can then test out the flowchart interactively and modify the dependency matrix if necessary. We show how the system is used to improve existing flowcharts as well as provide dynamic flowcharts that automatically rearrange the order of questions in response to changes in problem frequency.

7.1 Optimizing An Existing Flowchart



Leaves t Questions o	show_ bm_specs	show_ bm_contact_info	goto bm_3	show _swl2	goto bm_2_scan _glass_autofeed	done	show _swl2	goto bm_2_copy _scan_LED_error
(1) How can we help you?	Information	Information	Problems	Problems	Problems	Problems	Problems	Problems
(2) Do you need?	Specs	Contact_info	Problem Determination	Problem Determination	Problem Determination	Problem Determination	Problem Determination	Problem Determination
(3) What kind of problem are you having?	*	*	Scanning	Scanning	Scanning	Copying	Copying	Copying
(4) Does your scan complete without failure or error?	*	*	Yes	No	No	*	*	*
(5) Does the item you are copying scan in?	*	*	No	No	No	Yes	Yes	Yes
(6) Is there an error on LED or Panel?	*	*	*	Yes	No	*	*	*
(7) Does LED read PS/Ethernet or Token Ring?	*	*	*	*	*	Yes	Yes	No
(8) Does the printer print?	*	*	*	*	*	Yes	No	*

Figure 3: Converting the flowchart to a dependency matrix.

We begin by showing how the system is used to improve existing flowcharts. Figure 2 shows a small piece of a flowchart used for printer diagnostics. As described above, this flowchart can be converted directly into a dependency matrix, shown in Figure 3. Then

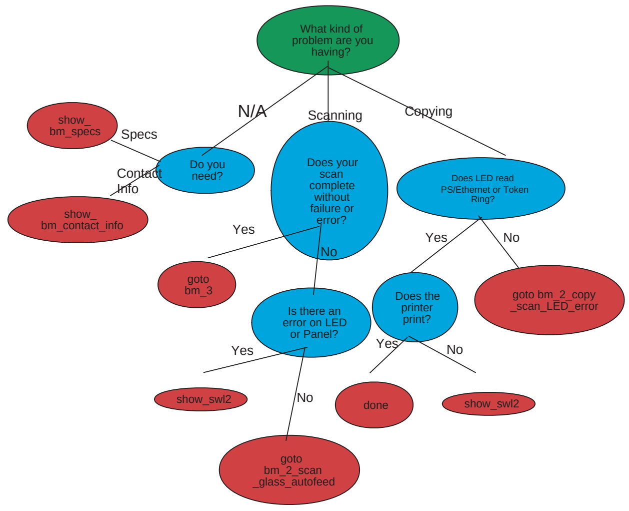

What kind of problem are you having?
N/A
Scanning
Copying
show_ bm_specs
Specs
Do you need?
Contact Info
show_ bm_contact_info
Does your scan complete without failure or error?
Does LED read PS/Ethernet or Token Ring?
Yes
No
Yes
No
goto bm_3
goto bm_2_copy _scan_LED_error
Is there an error on LED or Panel?
Does the printer print?
Yes
Yes
No
show_swl2
No
done
show_swl2
goto bm_2_scan _glass_autofeed

Figure 4: Converting the dependency matrix to an optimized flowchart.

the greedy algorithm is used to convert the dependency matrix into a new, optimized, flowchart, shown in Figure 4.

The new flowchart contains only 6 tests, compared with 8 tests in the original flowchart. It is easy to see that, assuming all states are equally likely and all tests are equally costly, the expected cost of diagnosis using the new flowchart is 2.5 tests, compared with 4.375 tests in the original flowchart, a savings of 43%. These numbers indicate roughly the kinds of improvements that can be obtained with legacy flowcharts.

7.2 Automatically Changing the Order of Questions

We now illustrate the advantage of generating flowcharts from dependency matrices by showing how the order of questions in the flowchart automatically changes to reflect changes in information, such as probability of different symptoms or problems. In contrast, a manually authored flowchart cannot be easily changed in this way.

7.2.1 Example - Automobile Fault Diagnosis

Our example is a simplified version of a real-life situation - diagnosing an automotive problem based on information about an unusual smell. First assume that drivers are able to faultlessly detect the exact smell. For example, if a fuel-injection problem occurs, generating a rotten-eggs smell, the driver does not confuse this with the sulfur-like smell generated by leaking gear lubricant. Under this assumption the smell is almost completely diagnostic of the problem. However there are two different states that have the same "Maple Syrup" smell. In this case additional information, such as whether the smell occurs primarily inside or outside the car, is needed to resolve the problem. The complete dependency matrix is shown in Table 1.

Given a prior distribution over the states, a flowchart is then generated using the *GREEDY* algorithm described in section 5. If a uniform prior is assumed, the resulting flowchart is the one shown in Figure 5. The flowchart first asks "What does it smell like?". Only if the smell is "Maple Syrup" is another question needed, "Where do you smell it?". The third question, "When do you smell it?" is not needed.

Note from the flowchart that if "Maple Syrup" is smelled both inside and outside the car, the diagnosis is "Radiator Leaking Coolant" - the rightmost node in Flowchart1. This is an example where multiple states are possible, but no further questions are available to distinguish between them - the **most likely state** is shown. Clicking on any node in the flowchart shows the state probabilities at that node. For example for the rightmost "Radiator Leaking Coolant" node there is a 2/3 probability that the problem is "Radiator Leaking Coolant" and a 1/3 probability that the problem is "Bad Heater Core". This reflects the fact that if 'Maple Syrup" is smelled both inside and outside the car it is twice as likely (according to the dependency matrix) that the cause is "Radiator Leaking Coolant" rather than "Bad Heater Core".

7.2.2 Scenario 1 - Changes in Symptom Probabilities

The assumption of "perfect smelling" is obviously unrealistic. For example, many drivers may confuse a smell of rotten eggs with a smell of sulfur. Table 2 shows a more realistic dependency matrix that reflects this - the true smell is the most likely to be reported, but it may be confused with similar smells.

The flowchart generated by the greedy algorithm no longer asks "What does it smell like?" first. It first asks "**Where** do you smell it?", and the next question changes depending on the answer to the first question - see Figure 6. If the smell is inside the car, the second question is "**What** does it smell like?", but if it is both inside and outside then the most-informative question to ask next is "**When** do you smell it?" This flexible modification of the question depending on the answers to previous questions illustrates the advantages of a learning approach over an expert's hard-coded flowchart. Note that Figure 6 does not show the complete flowchart - expanding the "What does it smell like?" node is needed, but results is a rather large and messy diagram.

7.2.3 Scenario 2 - Changes in State Priors or Test Costs

If the state priors change - for example a shipment of faulty gear lubricant results in a sudden rise in the occurrence of leaking gear lubricant problems - the flowchart will automatically respond by optimizing the order of the questions, and so the question "When do you smell it?", which is highly diagnostic of leaking gear lubricant, will be placed earlier in the flowchart, enabling the correct diagnosis to be made as quickly as possible. Similarly, if different tests have different costs and a previously expensive test becomes cheaper, it will rise earlier in the flowchart. Thus the flowchart dynamically takes into account all the relevant information to optimize the overall cost of diagnosis.

7.3 Exploration of New Questions

When a new problem or state appears it is sometimes the case that the new state will have symptoms that exactly match that of an existing state, but with one new differentiating factor. For example, a new device driver may have been released that has an obscure bug that causes the same set of symptoms as another known problem with the device. The new, as yet unknown, variable is the device driver level. In adding the new state, we would add a new question asking for the device driver level. In a large and complicated matrix, specifying the answer to this question for the two states we must differentiate is easy, but specifying the answer for all states may be tedious, and in fact a highly trained member of the support staff may not always be able to come up with such answers, or have the time to do so. When we add such a question q, it starts out offering very little information gain, since it can only be used to differentiate between two states. The trick is to occasionally do intelligent exploration to see if q is relevant to other states. A principled approach is to occasionally ask q in cases where we already have a state signature that is close to, but not the same as that of the states for which the answer for q is known. It is also possible to introduce a policy where q is asked only once a given problem is resolved and so not burden problem diagnosis too directly - such a policy may be viewed more favorably by customers. There are a number of reasonable prospecting strategies, and this general problem has been extensively studied, for example [1] and [3].

7.4 Weighting of Historical Data

Another important issue is the fact that older historical data on the frequency of state or symptom occurrence is typically less valuable than more recent data. Thus it is generally a good idea to put an aging policy in place that decays the value of older information, however the correct decay function is likely to be

States *Tests*	**Fuel Injection Problem**	**Mildew in A/C Evaporator**	**Gear Lube Leaking**	**Radiator Leaking Coolant**	**Bad Heater Core**
What does it smell like?	Rotten Eggs	Dirty Socks	Sulfur	Maple Syrup	Maple Syrup
Where do you smell it?	Both Inside and Outside Car	Inside Car	Both Inside and Outside Car	Outside Car 0.8 Both Inside and Outside Car 0.2	Inside Car 0.9 Both Inside and Outside Car 0.1
When do you smell it?	Engine is Running	Engine is Running	All the time	Engine is Running	Engine is Running

Table 1: DM1: Dependency Matrix for Automobile Diagnosis Example.

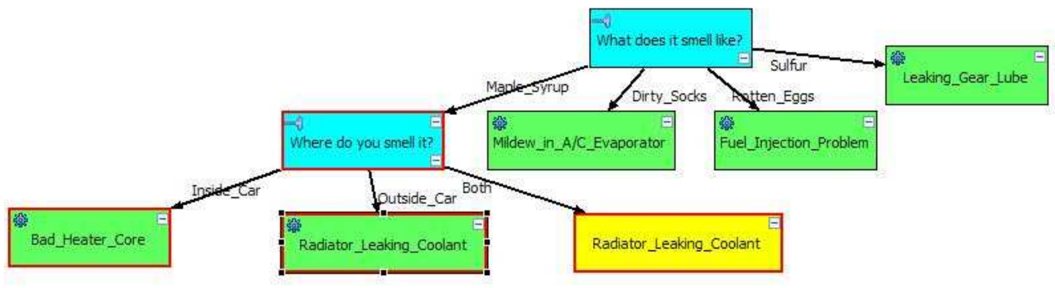


Figure 5: Flowchart1 - Automobile Diagnosis Example.

States *Tests*	**Fuel Injection Problem**	**Mildew in A/C Evaporator**	**Gear Lube Leaking**	**Radiator Leaking Coolant**	**Bad Heater Core**
What does it smell like?	Rotten Eggs 0.5 Sulfur 0.4 Dirty Socks 0.1	Dirty Socks 0.5 Rotten Eggs 0.2 Sulfur 0.3	Sulfur 0.5 Rotten Eggs 0.4 Dirty Socks 0.1	Maple Syrup	Maple Syrup
Where do you smell it?	Both Inside and Outside Car	Inside Car	Both Inside and Outside Car	Outside Car 0.8 Both Inside and Outside Car 0.2	Inside Car 0.9 Both Inside and Outside Car 0.1
When do you smell it?	Engine is Running	Engine is Running	All the time	Engine is Running	Engine is Running

Table 2: DM2: Dependency matrix for Scenario 1.

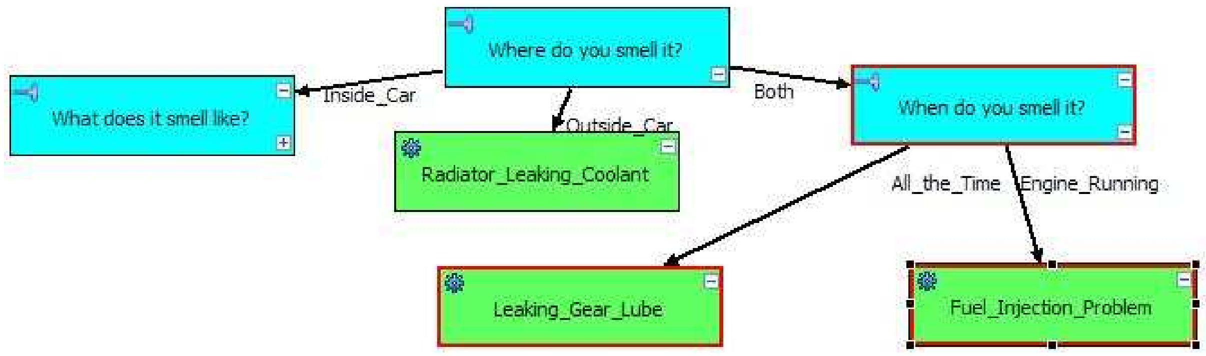


Figure 6: Flowchart2 - Flexible Question Ordering.

application-specific.

For example, a typical help center often faces the following problem: occasionally they get a flurry of calls about a single problem, say the call center is for internal IT problems and suppose a server is down somewhere. In these instances, until the problem is resolved, it may be desirable to ask the single question, "Does your problem have anything to do with such and such a server?"

7.5 When Flowcharts get too Big

Another area of interest to practitioners is when to break flowcharts up into smaller flowcharts. It is often not the most practical or efficient arrangement to have one big flowchart containing all states and questions. More typically in call centers, the call taker will talk to the caller briefly and may navigate a small initial flowchart to determine the basic nature of the problem in order to find another flowchart more geared to the problem at hand. Knowledge authors are therefore interested in automatic procedures for breaking flowcharts into smaller flowcharts.

8 Questions for Future Research

Two formulations of our problem admit a treatment via reinforcement learning[1]. The first is where for any given state there is a completely deterministic answer to each question. In this instance the objective is to minimize the expected number of questions to reach diagnosis. The second case is where we declare some ϵ tolerance and proceed to ask questions, some perhaps repeatedly, until one state remains with a probability of occurrence of at least $1 - \epsilon$. The simplest objective function (again to be minimized) is the expected number of questions to reach an ϵ-diagnosis. It will be interesting to see the degree to which reinforcement learning changes the ordering of questions in practical examples, and also the size limit of flowcharts to which reinforcement learning is practical.

The possibility of asking questions multiple times in a help-desk or self-help environment poses certain dilemmas. On the one hand there may just be one answer in the case at hand, and asking the question repeatedly does not mean getting additional information (and in fact can insult the caller). However, the question itself may be a type of probe which at any point in time has an uncertain answer. For example if we are diagnosing automobile problems, the state is a failing head gasket, and the probe is for engine temperature, although very likely, elevated engine temperature can either be present or not at any point in time and repeated sampling refines the diagnosis.

The problem of when to terminate a diagnosis should be amenable to more precise analysis. The notion of an ϵ-stopping point is obviously a simplification. What is the right ϵ and should there be the same ϵ for all paths? One should clearly consider the cost of a misdiagnosis, which is a function of both what misdiagnosis is given plus what the underlying correct diagnosis is.

Another interesting problem that arises in practice is that when traversals of the flowchart are recorded, we often do not get a clear indication of when a problem has been resolved. We get a record of what questions were asked and how they were answered, but often the questions stop at some point short of the end, maybe because the call-taker and/or end-user already realize what the problem and solution are. Our FLOAT implementation provides a convenient back-up mechanism enabling users to back up and change answers. What is the best way to incorporate this partial information? The approach we envisage is to take a partial traversal to be a "vote" for that subset of the tree, with the current marginal probabilities. When a user backs up to change the answer of a prior question, we entirely discount the earlier set of answers.

[1]Approaching our problem via reinforcement learning was suggested to us by Professor Doina Precup of McGill University.

References

[1] D. A. Berry and B. Fridstet. *Bandit Problems: Sequential Allocation of Experiments.* Chapman and Hall, New York, 1985.

[2] L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is np-complete. In *Information Processing Letters*, pages 15–17, 1976.

[3] M. K. Katehakis and H. Robbins. Sequential choice from several populations. *Proceedings of the National Academy of Sciences*, 92:8584–8585, September 1995.

[4] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S Stolfo. A coding approach to event correlation. In *Proceedings of the Fourth International Symposium on Integrated Network Management*, pages 266–277, 1995.

[5] S. R. Kosaraju, T. Przytycka, and R. Borgstrom. On an optimal split tree problem. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, pages 157–168, 1999.

[6] H. C. Ozmutlu, N. Gautam, and R. R. Barton. Zone recovery methodology for probe-subset selection in end-to-end network monitoring. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*, 2002.

[7] I. Rish, M. Brodie, and S. Ma. Accuracy versus efficiency trade-offs in probabilistic diagnosis. In *Proceedings of AAAI-02*, 2002.

[8] D. Sieling. Minimization of decision trees is hard to approximate. In *IEEE Conference on Computational Complexity*, pages 84–92, 2003.

[9] A. Zheng, I. Rish, and A. Beygelzimer. Efficient Test Selection in Active Diagnosis via Entropy Approximation. In *Proceedings of UAI-05*, 2005.