To appear in *Proceedings of the Fifth International Workshop on Principles of Diagnosis* (1994). New Paltz, NY.

Theory Revision in Fault Hierarchies

Pat Langley[*], **George Drastal, R. Bharat Rao and Russell Greiner**
Learning Systems Department, Siemens Corporate Research
755 College Road East, Princeton, NJ 08540 USA

Abstract

The fault hierarchy representation is widely used in expert systems for the diagnosis of complex mechanical devices. On the assumption that an appropriate bias for a knowledge representation language is also an appropriate bias for learning in this domain, we have developed a theory revision method that operates directly on a fault hierarchy. This task presents several challenges: A typical training instance is missing most feature values, and the pattern of missing features is significant, rather than merely an effect of noise. Moreover, the accuracy of a candidate theory is measured by considering both the sequence of tests required to arrive at a diagnosis and its agreement with the diagnostic endpoints provided by an expert. This paper first describes the Δ algorithm for theory revision of fault hierarchies that was designed to address these challenges, then discusses its application in knowledge base maintenance and reports on experiments that use Δ to revise a fielded diagnostic system.

1 Knowledge Maintenance and Machine Learning

Over the past decade, large-scale expert systems have found widespread use. However, developers have found that the cost of maintaining a knowledge base, over its lifetime, can be as high as the initial cost of its development. One response is to use machine learning to correct the knowledge base as problems emerge. Unfortunately, standard induction methods seem ill-suited to this task, as they are designed to use training data to construct a knowledge base *from scratch*, and the rate at which training data is generated by field users is typically too low to support regeneration of the knowledge base each time a revision is needed. As domain knowledge is available (in the form of the knowledge base), it makes sense to take advantage of this information (even if imperfect) to bias the induction process. Techniques for *theory revision* are suited for the task of knowledge maintenance, since they use training data to improve an initial domain theory.

[*]Current address: Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305

In this paper, we report research on theory revision that has been driven by our experience with CTX, a fielded expert system (now undergoing beta testing by 65 field service engineers in the U.S.) that diagnoses faults in the high-voltage generator of a computerized tomography (CT) scanner manufactured by Siemens Medical Systems. Although initial feedback indicates that CTX reduces both the number of repeat visits by field service engineers and the number of times they must consult a domain expert, users have reported errors in the knowledge base. In addition, we know that bug reports will accrue as the failure distribution drifts, due both to aging of the devices and device upgrades. These events create a need for periodic updating of the knowledge base to maintain an acceptable performance level. Our goal is to develop a theory revision system that can correct these errors in a cost-effective manner, by minimizing the need for manual revision by a knowledge engineer.

The task raises two problems that previous research in theory revision has not addressed. First, the diagnostic knowledge base takes the form of a *fault hierarchy*, which differs from the standard Horn clause representation used in most theory revision work. As domain experts and users seem comfortable with the representational bias provided by a fault hierarchy, we are reluctant to change to a different formalism. Second, most work on induction has assumed that all (or at least most) features appear in training and test cases. However, the selective nature of the diagnostic process means that only a few of the many possible feature values are specified in any instance, and the set of features present is correlated with the diagnosis itself (Rao, Greiner, & Hancock, 1994). We must deal with both issues as part of a practical approach to knowledge base maintenance.

In the next section, we explain the structure of fault hierarchies, examine the procedure by which CTX uses

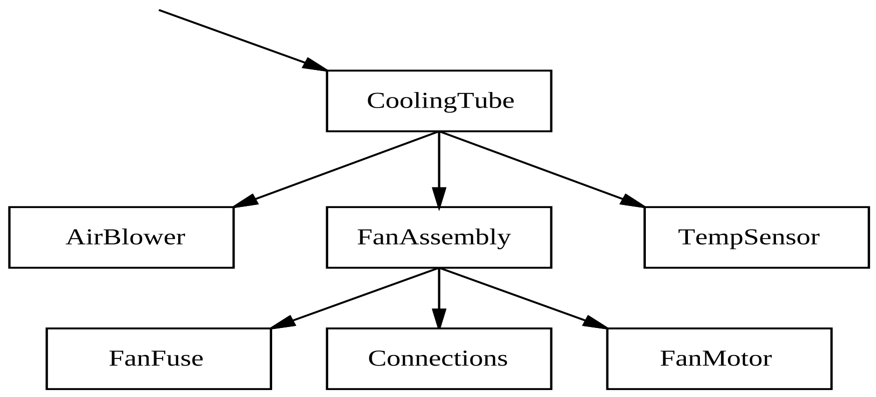

Figure 1: A partial fault hierarchy for the cooling subsystem of the X-ray tube in a computerized tomography scanner. Test and repair information is omitted.

such hierarchies to diagnose faults, and introduce the Δ algorithm, which revises an initial fault hierarchy in response to logs of diagnostic sessions. In Section 3 we present some hypotheses about the learning system's behavior, consider the experimental approach we have taken to test them, and report the results of our experiments. In the final section, we compare Δ with other approaches to theory revision, outline some directions for future work, and review the contributions of our research.

2 Theory Revision for Diagnostic Expert Systems

2.1 Representation of Diagnostic Knowledge

One representational formalism that has been used in diagnostic expert systems is the *fault hierarchy*. This structure is a directed acyclic graph (DAG) in which nodes correspond to possible faults in the device, and each child node corresponds to a possible cause of its parent node. Note that a node represents a functional failure, which need not correspond to a physically distinct component but rather to a set of components that is causally linked to that function. Figure 1 shows the portion of the CTX fault hierarchy that deals with the X-ray tube cooling subsystem in a CT scanner. This hierarchy states that a malfunction in CoolingTube can be caused by either a fault in the AirBlower, the FanAssembly, or the TempSensor. Similarly, a malfunction in the FanAssembly can be caused by a blown FanFuse, loose Connections, or a faulty FanMotor.

Each node N has an associated "node test", written T_N, for determining the presence of its specified fault. Each such node test is a Boolean combination of a set of primitive tests (e.g., *temperature* $> 80 \wedge$ *loose-connection* $=$ *true*). In each situation, the node test will return either 'true' or 'false', which (respectively) confirms or disconfirms the presence of the specified fault. We will refer to a test result that confirms node N as "$\mathcal{T}_N$" and to one that disconfirms N as "$\overline{\mathcal{T}}_N$". (Hence, $\mathcal{T}_{FanFuse}$ means the test associated with the "FanFuse" node, $T_{FanFuse}$, was confirmed, and $\overline{\mathcal{T}}_{FanFuse}$ means this test was disconfirmed.) In addition to a test, each terminal node N in the fault hierarchy specifies the repair $\mathcal{R}_N$ intended to correct its associated fault. For example, the 'FanFuse' node in Figure 1 has an attached "replace the fan fuse" repair (denoted $\mathcal{R}_{FanFuse}$) that will, when appropriate, eliminate the higher-level faults in the fan assembly and tube cooling functions.[1]

Typically, nodes higher in the fault hierarchy represent faults in larger modules or subsystems of a device, which can be caused by any of the more localized faults below them. Thus, elimination of a high-level fault, N, implies the elimination of all faults that fall below N in the hierarchy (unless one can reach the lower nodes through another path[2]). The fault hierarchy used in the fielded CTX system contains approximately 580 fault nodes and 400 different tests that recommend one of 150 alternative repairs.

2.2 Diagnosis Using a Fault Hierarchy

The CTX system is implemented in TestBench,[3] a diagnostic shell that evaluates a fault hierarchy by depth-first traversal of the DAG, beginning at a root node R, where $\mathcal{T}_R$ is an initial (i.e., presenting) symptom, which is known to be true. On reaching the node N, TestBench invokes (or directs the user to carry out) the test T_N associated with N. If the test result confirms that fault N is present in the device, TestBench examines N's children to determine which of these more specific faults is responsible for the problem, beginning with the leftmost child. Alternatively, if the test result disconfirms N, TestBench consid-

[1] Figure 1 does not explicitly show either the tests or the repairs associated with the nodes.

[2] This can occur when N has multiple parents, producing a hierarchy that is an arbitrary directed acyclic graph rather than a tree like the one in Figure 1.

[3] TestBench is a trademark of Carnegie Group, Inc.

ers the sibling fault node to N's immediate right. The diagnostic process halts when TESTBENCH either confirms a terminal node (and returns the associated repair), or when TESTBENCH disconfirms all children of a confirmed fault, in which case it gives up (and returns "$\mathcal{N}o$-$\mathcal{D}iagnosis$"). Thus, diagnosis takes the form of depth-first search with no backtracking.

For example, suppose we suspect that a fault may exist in the cooling system, under the "CoolingTube" node of the fault hierarchy is shown in Figure 1. TESTBENCH would first test for this fault, in this case asking the engineer to see if the X-ray tube is overheating. If true, this test confirms the fault and the system *tentatively hypothesizes* that the fault resides in the AirBlower, since this is the leftmost child of the CoolingTube node. If the test associated with this node returns false, then the AirBlower fault is disconfirmed and TESTBENCH then hypothesizes the next fault in this set of children, FanAssembly. If the test $T_{FanAssembly}$ (switching the fan on and noting if it does not rotate) succeeds, TESTBENCH confirms the FanAssembly fault and then considers the first of its children, FanFuse. If checking the fuses reveals no failure there, TESTBENCH disconfirms this hypothesis and considers its sibling, Connections. If its test succeeds, TESTBENCH suggests a repair ($\mathcal{R}_{Connections}$ = "replacing the connector") and the diagnostic process halts.

Note that any diagnostic session actually corresponds to a *path* through the fault hierarchy, with confirmed tests leading downward and disconfirmed tests leading to the right. For instance, the second of the scenarios we described above produces the path $\langle \mathcal{T}_{CoolingTube}, \overline{\mathcal{T}_{AirBlower}}, \mathcal{T}_{FanAssembly}, \overline{\mathcal{T}_{FanFuse}}, \mathcal{T}_{Connections} \rangle$, which indicates an alternation between movements downward and to the right through the fault hierarchy in Figure 1. The left to right ordering of sibling faults is a natural way to encode the preferences of an expert test engineer who is faced with the choice of alternate fault hypotheses to pursue. Notice also that TESTBENCH has performed only 5 of the 7 tests of this fault hierarchy.

A set of these diagnostic sessions, augmented with the correct repair and possibly other test values provided by the human expert, form the "labeled training instances" used by the Δ theory revision system, described below.

2.3 The Δ Theory Revision Algorithm

To revise an incorrect fault hierarchy, we considered first translating the initial fault hierarchy into an equivalent representation, such as Horn clauses or a decision tree, then using an existing induction method to modify this structure, and finally translating the result back into a revised fault hierarchy. However, we rejected this approach for two reasons. First, a minor revision in the search space of either Horn clauses or decision trees may correspond to a large step in the space of fault hierarchies — a step that may not preserve the original causal structure. Since any proposed revision will be subject to approval by domain experts, the allowable transformations should minimize violations of the causal structure in order to be comprehensible to them. Second, traditional induction methods typically assume that most feature values are present in the training data and that the hidden feature values are uncorrelated with the class. As noted earlier, our learning task clearly violates this assumption. Therefore, we have developed Δ, a theory revision system that operates directly with fault hierarchies and handles training data with missing features.

The Δ system uses four types of transformations to move through the space of fault hierarchies, each mapping one hierarchy H to a slightly different hierarchy:

- $Add_{P,C,i}(H)$ *adds a link to* H, by creating a new connection from node P to node C, making C the i^{th} child of parent P; this creates a new path through which TESTBENCH can reach C.
- $Delete_{P,C}(H)$ *deletes a link from* H, by removing the existing connection between a parent node P and one of its children C; this eliminates one of the paths to C.
- $Move_{P_1,P_2,C,i}(H)$ *moves one of* H*'s node*, by removing a node C from its parent P_1 and making it the i^{th} child of P_2; this is equivalent to deleting the link from P_1 to C and adding a link from P_2 to C.
- $Switch_{P,C_1,C_2}(H)$ *switches two of* H*'s nodes*, by taking two nodes, C_1 and C_2, with a common parent P and interchanging their positions; this alters the order in which TESTBENCH considers C_1 and C_2 after confirming P. Notice this is equivalent to (at most) two *Move* transformations.

For each transformation, $\sigma \in \{$*Add*, *Delete*, *Move*, *Switch*$\}$, we let $\sigma(H)$ represent the hierarchy formed by applying σ to H. For example, using the hierarchies shown in Figure 2, $Switch_{S,A,B}(KB_I) = KB_T$; notice also that $Move_{S,S,A,2}(KB_I) = KB_T$.

The Δ system places some restrictions on the application of these transformations. In particular, it forbids deletions of links that would completely disconnect one or more nodes from the hierarchy, and it disallows transformations that would introduce loops by making a node its own descendant. Nevertheless, there is a very large space of possible structures that the transformations can generate from the initial fault hierarchy.[4] For example, given a complete fault hierarchy with depth d and b branches at each level, there are $O(b^{2d})$ possible transformations.

[4]Note that these transformations only alter the structure of the hierarchy; they do not modify the primitive tests associated with each node, which for now we assume to be correct. Section 4.2 explains why this is not a restriction.

The Δ algorithm employs a simple hill-climbing strategy to search this space, using the given initial fault hierarchy as its starting point. On each cycle, Δ applies each of the transformations $\{\sigma\}$ to the current hierarchy H in all legal ways to generate a set $Neighbors(H) = \{\sigma(H)\}$ of revised hierarchies, each differing from H by a single modification. It then uses a given set of labeled training examples $S = \{s_j\}$ (described above) to evaluate the empirical accuracy of H and each $H' \in$ Neighbors(H): That is, TestBench uses the performance component described in Section 2.2 to execute each hierarchy H_i on a training example s_j, which either returns some repair or fails, returning "$\mathcal{N}o$-$\mathcal{D}iagnosis$". Δ gives H_i a score of 1 if its repair on s_j corresponds to the correct repair (which labeled the example), and a score of 0 otherwise. H_i's empirical accuracy is then the average of these scores over the set S.[5]

The system then selects the most accurate knowledge base from Neighbors(H), which we denote H^*. If H^* is more accurate than the current hierarchy H, then Δ replaces H with H^* and iterates: seeking a neighbor $H^{**} \in$ Neighbors(H^*) with a yet higher accuracy score, and so forth. Otherwise, if none of H's neighbors has a better score than H, Δ halts and returns H as the best of the fault hierarchies it has encountered.

We overlay a bias in favor of minimal change upon this simple scheme as follows. Transformations are applied first to leaf nodes of the DAG before they are tried at successively higher levels, and ties in the evaluation function are broken by choosing the earlier revision. As typical fault hierarchies contain fewer initial symptoms than diagnostic endpoints, and relatively few nodes have multiple parents, nodes towards the leaves are traversed less frequently than nodes nearer the root, which means changes to the lower nodes (closer to the leafs) will affect fewer instances than changes to the upper nodes. This bias implements a preference for retaining as much as possible of the original fault hierarchy, which is reasonable since we expect that any serious errors in the hierarchy would elicit a correspondingly large number of bug reports. Similarly, we apply a preference ordering to the four transformations that reflects the behavior of human domain experts in the task of theory revision, based on our observations.

3 Experimental Evaluation of the Δ Method

For the users of Δ to consider the system successful, it must revise fielded knowledge bases (such as CTX) to reduce the number of subsequent bug reports. However, as we do not yet have access to these reports, we have resorted to other techniques to obtain a preliminary evaluation. In this section, we report some initial experimental studies using a data set that was synthesized by introducing plausible errors into a particular knowledge base which, for purposes of the study, we assume to be correct. The rest of this section presents the dependent measures and independent variables of interest, along with some hypotheses about Δ's learning behavior. We then describe the nature of the training and test sets used in our studies. Finally, we report the outcome of our experiments and compare these results to our hypotheses.

3.1 Experimental Variables and Hypotheses

Three main factors should influence the behavior of Δ. The first is the number of training cases the learning system has observed. Although Δ is nonincremental, we can still generate 'learning curves' by measuring the performance of the revised fault hierarchy after every n training instances. This will give us information about the rate of learning and its asymptotic performance.

A second central factor is the target fault hierarchy. We will not vary this term in our experiments, as our main aim is to evaluate Δ's ability to acquire the correct TestBench hierarchy for the CTX domain. For the purpose of these experiments, we assume that the fielded knowledge base is the target, and furthermore that its structure is representative of TestBench fault hierarchies for other applications.[6] Our experiments were performed using a connected subgraph of the full fault hierarchy which includes 63 unique tests that are organized into 66 failure nodes, with 39 possible repairs reachable through 41 alternative paths. This size is typical of a connected subgraph in the CTX fault hierarchy.

The distance between the initial and target hierarchy is the third important factor. We generate an initial fault hierarchy by applying inverses of the learning transformations from Section 2.3 to the target hierarchy, varying the distance by introducing different numbers of such 'bugs'. Figure 2 gives a simple example in which the switch operator transforms the target hierarchy into an initial one. In this case, the distance is one, but use of multiple transformations can produce initial hierarchies more distant from the target.[7] Based on inspection of the development history of the CTX system, we have selected ten bugs that appear typical, including four examples of node movement and two cases each of switching nodes, adding a link, and deleting a link. We combine these to generate different experimental starting points.

[5]This is actually a simplified description of the process; Section 3.2 discusses some other complications. Also, Section 4.2 suggests a more efficient implementation.

[6]In a companion corpus of experiments, we used a different target fault hierarchy, taken from a different fielded application. Those results are not reported here as they are essentially the same as the ones described below.

[7]Because different sets of transformations can produce the same fault hierarchy, we compare the mutated hierarchy with the target to compute the minimum distance; we use this measure of distance in our experiments.

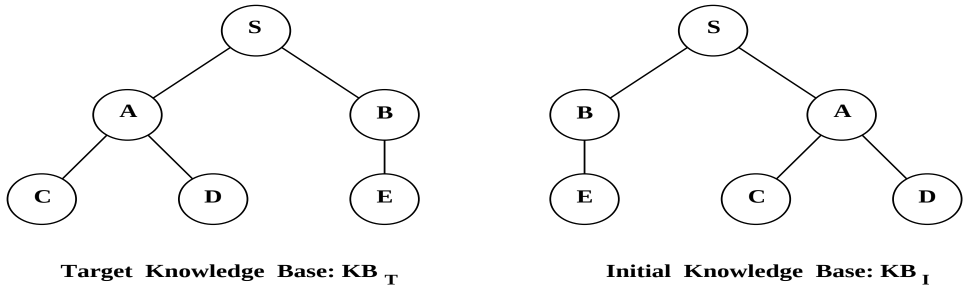

Figure 2: The learning transformations from Δ can generate an initial TestBench hierarchy from the target hierarchy. Here switching nodes transforms the target fault hierarchy, KB_T, which tests `A` before `B`, into the "initial" hierarchy, KB_I which performs these tests in the reverse order.

The goal of Δ is to revise the initial fault hierarchy in ways that improve its ability to propose correct repairs. This suggests diagnostic accuracy as the natural performance measure in our studies. However, we would also like Δ to carry out an efficient search of the space of revisions, which suggests the number of revision steps during learning as a second dependent measure.

These independent and dependent variables suggest three hypotheses about the behavior of Δ, each of which seems desirable for a theory revision system. First, the accuracy of the revised knowledge base should increase monotonically with the number of training instances, as with most induction techniques. More important, the number of training cases required to reach asymptotic accuracy should grow only linearly with the distance between the initial and target hierarchies. Finally, the number of revision steps that Δ takes during learning should scale well (e.g., linearly) with the distance between the initial and target hierarchies, given a sufficient number of training instances.

3.2 Generation of Training and Test Data

Although we must use artificial data to evaluate Δ, we would like our instances to realistically simulate bug reports from the field. Here we assume that a field engineer performs diagnostic tests as specified by the initial TestBench hierarchy and, if the suggested repair does not solve the problem, the engineer consults the domain expert, who recommends additional tests that lead to the correct repair. We will assume that the expert's suggestions are always correct.

Hence, a bug report includes a transcript of the test values elicited by following the initial hierarchy, any additional test values given by the expert, and the correct diagnosis. Transcripts of cases that were *correctly* diagnosed by the initial hierarchy are also used by the learning system. Note that in both cases, the training instance includes only some of many possible tests, and the ones included are determined by both the initial hierarchy and the expert. This feature distinguishes our work from other research on theory revision.

To generate data of this form, we consult the target TestBench hierarchy, which here plays the role of the domain expert. The first step in generating a training instance involves randomly selecting some path through the target hierarchy. This provides a sequence of tests and their associated values, along with the correct repair. For example, consider again the target hierarchy in Figure 2, which contains three such paths: $\langle \mathcal{T}_A, \mathcal{T}_C \rangle$ $\langle \mathcal{T}_A, \overline{\mathcal{T}}_C, \mathcal{T}_D \rangle$, and $\langle \overline{\mathcal{T}}_A, \mathcal{T}_B, \mathcal{T}_E \rangle$. Note that these paths specify the tests that the *correct* TestBench hierarchy would generate. For a given situation, however, the initial hierarchy may propose irrelevant tests (i.e., tests whose values are *not* needed by the target hierarchy) and omit relevant ones (i.e., tests whose values *are* needed by the target hierarchy). For each irrelevant test value requested by the initial hierarchy, Δ inserts a random value into the training case, and for each omitted relevant test, Δ inserts the value corresponding to the correct path, to simulate advice given by the expert.

For example, consider the first path above, $\langle \mathcal{T}_A, \mathcal{T}_C \rangle$. When using the problematic initial hierarchy KB_I, shown on the right of Figure 2, TestBench would consider the node B (and possibly E) before considering A and C. Notice however that the target hierarchy KB_T only provides test values for T_A and T_C, but not for either T_B nor T_E. We therefore consider all possible values of T_B and T_E consistent with the performance element's traversal of the initial hierarchy; *viz.*, i_1:$\langle \mathcal{T}_B, \mathcal{T}_E, \mathcal{T}_A, \mathcal{T}_C \rangle$, i_2:$\langle \mathcal{T}_B, \overline{\mathcal{T}}_E, \mathcal{T}_A, \mathcal{T}_C \rangle$, and i_3:$\langle \overline{\mathcal{T}}_B, \mathcal{T}_A, \mathcal{T}_C \rangle$. These training instances cause the performance element to return diagnosis E, $\mathcal{N}o\text{-}\mathcal{D}iagnosis$, and C, respectively. Case i_1 simulates a bug report in which the expert system's advice leads to an incorrect diagnosis.[8] Case i_2 simulates a report of the expert system failing to arrive at a diagnosis because the confirming test $\mathcal{T}_B$ is a red herring. Case i_3 leads to the correct diagnosis, though an unnecessary test T_B is ordered. Δ

[8]It is easy to imagine how $\mathcal{T}_E$ could confirm a fault in E even though E is not actually faulty. If E were dependent on C, then T_E alone might never have been intended to discriminate E from C, unless T_E were preceded by a test to first rule out C.

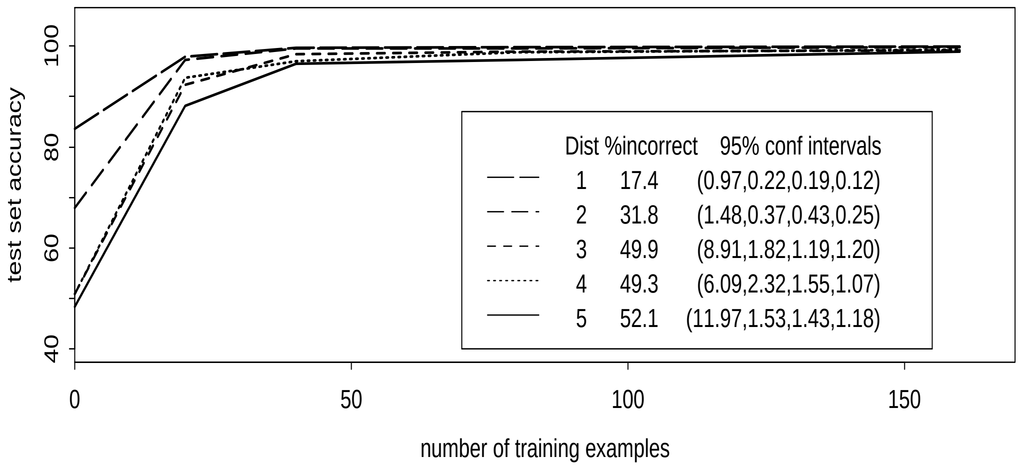

Figure 3: Diagnostic accuracy of the revised hierarchy as a function of the number of training instances, for each distance between the initial and target hierarchies.

will stochastically select one of these (augmented with the correct repair and the other relevant tests), with a probability that depends on the unspecified tests that had to be performed, and return some specific value.

The above description explains our approach to generating a training case from an individual path through the target hierarchy. To generate an entire training set, we must also use some distribution of paths. Here, we assume paths are uniformly distributed, and we select them randomly with replacement. For the target hierarchy in Figure 2, we would select each path with $\frac{1}{3}$ probability independently. Since in general a hierarchy is a DAG rather than a tree, this strategy produces some repairs more frequently than others. This seems a plausible assumption in the absence of an actual fault distribution.

In generating test data, many of the same issues arise as for training instances. We randomly select a path from the target hierarchy and, for each test along this path, we include the value needed to continue toward the selected repair. However, instead of inserting random values for only those tests that the initial hierarchy would request, we insert random values for all tests not along the correct path. For example, one possible test case from the above path would be $\langle \mathcal{T}_A, \overline{\mathcal{T}}_B, \mathcal{T}_C, \mathcal{T}_D, \overline{\mathcal{T}}_E \rangle$. This scheme ensures that any revised hierarchy can access any test during the diagnostic process. Naturally, we assume the test cases follow the same distribution as the training instances.[9]

3.3 Experimental Results

Using the framework described above, we carried out an experiment designed to test our hypotheses. We varied the number of training instances available to Δ from 20 to 160 and the distance between the initial and target hierarchy from one to five, in each case measuring the two dependent variables discussed earlier. For each condition, we averaged the results over 10 to 30 different initial hierarchies and 10 to 30 different training sets, both randomly generated.

Figure 3 plots the accuracy of the revised knowledge base on 500 test instances (the same test set for each run) as a function of these two factors. As expected, the accuracy of the revised hierarchy increases monotonically with the number of training instances (shown for 20, 40, 80, 160), approaching 100%. More interesting, the number of instances required to reach this level increases roughly linearly with the distance between the initial and target hierarchies, as proposed in our second hypothesis. The legend in Figure 3 shows, for each distance value, the average percentage of total training instances that are misclassified by the initial hierarchy, and 95% confidence intervals on each point.

The results for our second dependent variable, the number of steps occurring during theory revision, appear in Figure 4. This graph is generally consistent with our third hypothesis, showing that Δ's number of revision steps grows approximately linearly with the distance from the initial hierarchy to the target. This relation seems to hold for all training set sizes greater than zero. At least for the distances examined in this study, Δ appears to scale well to increasing amounts of mutilation in the target TestBench hierarchy.

In summary, our experimental results have generally borne out the hypotheses we proposed earlier in the section. This suggests that Δ has the characteristics one desires from a theory revision system, and that its behavior will be robust even when given a large fault hierarchy that contains many errors and training instances that omit the values of many features.

[9]Since we know the distribution of synthetic data in these experiments, we could analytically compute the accuracy of a hypothesis. This will not be possible, however, when data are provided by the user population.

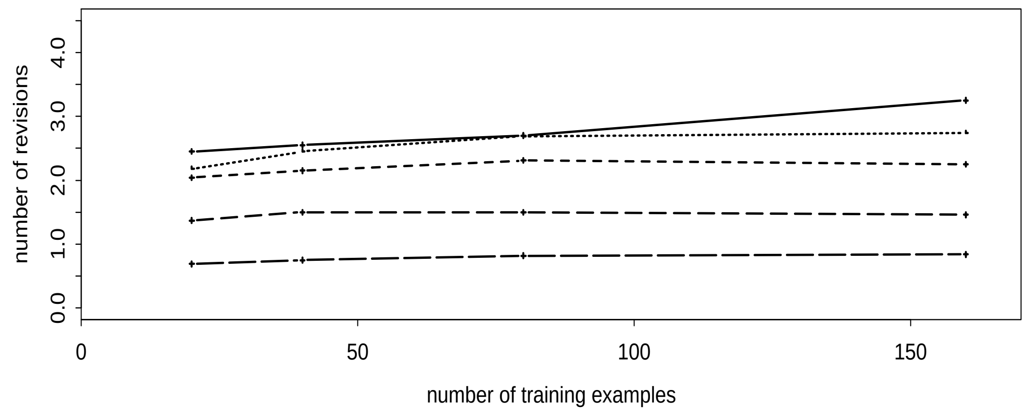

Figure 4: Revision steps during learning as a function of the number of training instances, for each distance between the initial and target hierarchies.

4 Discussion

4.1 Related Research on Theory Revision

The approach we have taken with Δ has close connections with other work on theory revision, including Ginsberg, Weiss, and Politakis (1988), Ourston and Mooney (1990), Craw and Sleeman (1990), Towell (1991), Cain (1991), Richards and Mooney (1991), Wogulis and Pazzani (1993), and Asker (1994). Starting from an initial domain theory obtained from experts, these methods also iteratively modify that theory to improve accuracy on a set of training cases. Like them, Δ uses the training instances to direct a non-incremental hill-climbing search through the space of domain theories.

However, the hypothesis space of fault hierarchies considered by Δ differs from the space searched by others, and the transformations used in Δ implement one-step revisions that may correspond to multiple-step revisions in other systems. The bias that is embodied in the concept description language of fault hierarchies and in the set of transformations is a familiar and intuitive one for experts in technical diagnosis. The bias toward using as much of the initial domain theory as possible, implemented in Δ by a preference ordering on transformations, can also be found in Drastal, Raatz, and Czako (1989).

One can also view incremental methods for the induction of decision trees (e.g., Schlimmer & Fisher, 1986; Utgoff, 1989) as carrying out a form of theory revision. Again, this approach performs a hill-climbing search using the current knowledge base as its starting point. The standard operators here involve extending the decision tree downward, pruning the tree, and reversing the order of two tests. Typically, this work assumes that one constructs a decision tree from scratch, but the basic approach should apply equally well when an expert provides an initial tree.

Our experimental studies have also drawn from the previous work on theory revision. In particular, Rose (1989) reports experiments that systematically vary the distance between the initial and target theories, using both accuracy and number of revision steps as dependent measures. In addition, Rose (1989), Ourston and Mooney (1990), and others have presented learning curves that measure accuracy as a function of the number of training cases seen by the theory revision system.

4.2 Directions for Future Work

Although our experiments with Δ have been encouraging, we have yet to explore many other facets of theory revision for expert systems. For example, we have focused on a particular (rather general) set of hierarchy–to–hierarchy transformations, which modify the "structure" of the hierarchy. There are other obvious transformations that affect the "contents" of the individual nodes. In particular, we could define transformations that change the test within a particular node; such an operator could transform the "Is the temperature above 70^o?" primitive test to, say, "Is the temperature above 80^o?". Although it would be easy to incorporate such transformations into Δ, we have not found them necessary, as almost all of the tests used in our applications are inherently binary, of the form "Is light no. 3 on?".[10]

Second, although our descriptions deal only with "connected hierarchies", where there is a path from the root to each leaf (repair) node, we could use the same Δ procedure on disconnected hierarchies. In particular, it could handle hierarchies that include both a connected component KB_{cc} (like the one shown in Figure 1) and a small collection of "extra nodes" that contain tests and/or repairs not included in the connected part (e.g., a "bad wire" node, with a test and repair not in KB_{cc}). Notice that Δ can use an add transformation to link one of these auxiliary nodes into the existing connected portion, which has the effect of

[10]In fact, we informally estimate that over 95% of our tests are inherently binary.

adding in a new test into that hierarchy. We can use this same "trick" to acquire new repairs, by adding a link from the some node in KB_{cc} to a node with a novel repair.[11]

A third extension concerns Δ's greedy approach, which currently generates and evaluates all possible revisions. Although this scheme is easy to describe, it is inefficient for large knowledge bases and large training sets. Even in our controlled experiments, Δ generated approximately $10,000$ neighbors for each current hierarchy. An alternative approach would generate possible revisions more selectively, based only on training cases that the current theory misclassifies. For example, such an algorithm would note when a desired test has been omitted on a training case, then consider moving the associated node to a position in which it would have been used in the case. Similarly, the method would consider removing only links to those nodes whose associated tests appear in a faulty diagnosis. We are currently testing these changes to Δ, and we expect them to speed the theory revision process with no loss of accuracy.

Finally, although we have emphasized the accuracy of diagnosis, we must also be able to use feedback on the sequence of tests requested. Incorporating a penalty for unnecessary tests into the evaluation function should be a straightforward way to direct the search toward theories that arrive at a correct diagnosis by the most preferred path.[12]

4.3 Contributions of the Research

In this paper, we described an approach to theory revision for diagnostic expert systems, and its implementation in Δ. This effort was motivated by the requirement to operate directly with a knowledge representation that has been used in diagnostic expert systems of large scale, and to meet acceptance criteria that include both the cost and accuracy of diagnosis. Our approach also had to be tolerant of missing feature values, and the method had to be able to produce highly accurate fault hierarchies without a large training set. Our initial experimental studies indicate that the Δ algorithm tends to converge rapidly on accurate knowledge bases, often reconstructing the actual target theory that was used to generate the training and test data. We have shown this using training sets containing randomized values for tests that are not essential to a given diagnosis. We also found that Δ scales well as one increases the distance between the initial and target hierarchy.

[11]Of course, we still require an expert to specify these auxiliary tests and repairs.

[12]Note that, in general, this "preference" is a function of many variables, including time and equipment needed to perform a test, risk of causing damage by testing, reliability of the test result, and information gain of the test.

References

Asker, L. (1994). Improving accuracy of incorrect domain theories. *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 19–27). New Brunswick, NJ: Morgan Kaufmann.

Cain, T. (1991). The *Ductor*: A theory revision system for propositional domains. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 485–489). Evanston: Morgan Kaufmann.

Craw, S., & Sleeman, D. (1990). Automating the refinement of knowledge-based systems. *Proceedings of European Conference on Artificial Intelligence* (pp. 167–172). Stockholm: Pitman.

Drastal, G., Raatz, S., & Czako, G. (1989). Induction in an abstraction space: A form of constructive induction. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 708–712). Detroit: Morgan Kaufmann.

Ginsberg, A., Weiss, S., & Politakis, P. (1988). Automatic knowledge base refinement for classification systems. *Artificial Intelligence*, *35*, 197–226.

Ourston, D., & Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 815–820). Boston: AAAI Press.

Rao, R. B., Greiner, R., & Hancock, T. (1994). Exploiting the absence of irrelevant information: What you don't know *can* help you. *Working Notes of the AAAI Fall Symposium on Relevance*. New Orleans: AAAI Press.

Richards, B., & Mooney, R. (1991). First-order theory revision. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 447–451). Evanston, IL: Morgan Kaufmann.

Rose, D. (1989). Using domain knowledge to aid scientific theory revision. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 272–277). Ithaca, NY: Morgan Kaufmann.

Schlimmer, J. C., & Fisher, D. (1986). A case study of incremental concept induction. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 496–501). Philadelphia: Morgan Kaufmann.

Towell, G. (1991). *Symbolic knowledge and neural networks: Insertion, refinement, and extraction.* Doctoral dissertation, Computer Sciences Department, University of Wisconsin, Madison.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning*, *4*, 161–186.

Wogulis, J., & Pazzani, M. (1993). A methodology for evaluating theory revision systems: Results with Audrey II. *Proceedings of Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1128–1134). Chambéry: Morgan Kaufmann.