Repair Strategies in a Diagnostic Expert System

Jeff Pepper and Gary S. Kahn
Carnegie Group Inc.
Pittsburgh Pennsylvania 15219

Abstract

Successful machine diagnosis consists not only of sound diagnostic reasoning, but also the selection of appropriate repairs, sequencing the repairs correctly, interactively validating the success of each repair, and performing follow-on diagnosis in case of repair failure. We discuss some of the issues involved in formulating a repair strategy for an expert system, review some aspects of expert human behavior with respect to repair strategy in a complex domain, and finally describe an implementation of a repair strategy in the TEST diagnostic shell.

1. INTRODUCTION

Diagnostic applications remain the most heavily explored area of expert systems technology, but little attention has been given to the selection, sequencing and interactive verification of repairs. These aspects of diagnostic behavior, collectively referred to in this paper as *repair strategy,* are critical to the success of a real-world machine diagnosis expert system.

Typically, repair strategy is seen as a straightforward selection keyed to a successful diagnostic conclusion ([Bennett 81,Bylander 83, Fink 84, Ali 85, Maletz 85, Strandberg 85] and others). In a few cases, supplementary repair or treatment modules have been developed to provide customized or special case recommendations [Clancey 84, Kahn 85]. However, apart from possibly [Hofmann 86], none of this work has attempted to formulate an approach which permits the degree of integration of diagnosis and repair tasks required to effectively troubleshoot faults in complex machines.

TEST (Troubleshooting Expert System Tool), as reported in this paper, recognizes a larger class of integration issues than earlier work, and provides a representation more easily customized to differences in repair strategy, even within the same application domain. TEST is an application independent diagnostic problem-solver which operates on domain-specific knowledge bases. Work on TEST began late in 1985, and there are now several applications in progress, the largest being Ford Motor Company's Service Bay Diagnostic System [Tedesco 86]. Other applications include factory floor machine diagnosis, online monitoring of generator equipment, and computer performance tuning. TEST has been implemented in Common Lisp using Knowledge Craft™ [Pepper 86a], and a C version is under development.

The next section describes ways in which human expert diagnosticians integrate diagnostic reasoning and repair strategy. The remaining two sections describe TEST'S repair strategy, and how it models aspects of this behavior.

2. WHAT MAKES REPAIR HARD

While it is easy to imagine a prototypical diagnostic system that identifies a single cause for observed symptoms, and recommends the corresponding, inevitably successful repair, this is an ideal but unlikely case. More often, technicians make repairs during the diagnostic process itself, and are prepared to (1) select from among competing repair alternatives, (2) delay making certain repairs until deeper causes are identified, and (3) continue with the diagnosis if the repair does not succeed.

Selection and sequencing. The selection of a repair for a single failed component depends on several factors: the cost of the repair, its degree of "goodness", and the availability of required tools and parts. But sometimes a diagnosis results in several failed components requiring repair, and sequencing becomes an issue. For example, a car's failure to start may be caused by a dead battery, which requires either recharging or replacement of the battery. However the diagnosis must proceed further to find the cause of the dead battery, possibly discovering a fault in the charging system. This second fault also has repairs associated with it. In such a case, the technician must determine the proper sequencing of repairs, and handle situations where performing one repair makes another one unnecessary. Further, repairs change the state of the UUT (unit under test), invalidating some or all evidence gained prior to the repair and complicating the follow-up diagnosis that must be performed if any repairs fail.

Handling failed repairs. Repairs fail for three reasons. First, the selected repair may be performed incorrectly because of errors in technique, use of the wrong replacement part, or use of a correct part which turns out to be defective. Second, the wrong repair may be selected because of an error in the technician's domain knowledge, or faulty evidence obtained during

diagnosis. And third, the diagnosis addressed by the repair may be incorrect. This last reason, misdiagnosis, can result from errors in domain knowledge or faulty evidence gained during diagnosis. Or a diagnosis may be partially correct, but the effectiveness of the repair masked by co-occurring problems that have yet to be identified. Until all causes for the failure are identified and repaired, the UUT remains in a failed condition.

Hypothesized diagnosis. Sometimes repairs are made on the basis of a hypothesized diagnosis, because it is difficult or impossible to acquire direct confirming evidence. The hypothesized diagnosis is then confirmed by the repair itself, or discarded in favor of a new diagnosis. For example, one car diagnosis heuristic states that *"to fix certain fuel pressure problems, replace the fuel pump. Afterwards, do a fuel pressure test, and if the test fails, conclude a bad fuel pressure regulator.* In this example, the conclusion is hypothesized, then either confirmed or replaced by another conclusion as a result of evidence obtained during the repair.

A hypothesized diagnosis can also result from a repair in cases where a previously confirmed diagnosis is proven to be correct but inadequate. As an example, another car repair heuristic states that *"for a no-start, if you find that the battery posts are corroded but cleaning the posts does not solve the problem, it must be due to a fault in the charging system".* In this case, the correct repair was made for a properly diagnosed condition, but performing the repair revealed that the condition was not severe enough to cause the problem.

3. TEST'S REPAIR STRATEGY

Because the repair strategies discussed in the previous section are an integral part of human expert diagnostic behavior, it is desirable to include similar capabilities in expert systems. In this section we provide an overview of TEST, and discuss its repair stratgies in detail. (More complete discussions of the TEST architecture are given in [Kahn 87a], [Kahn 87b], and [Pepper 86b].)

3.1. Overview of TEST

TEST provides a domain-independent problem-solver, together with a library of schematic prototypes which constitute the structure within which domain-specific knowledge bases must be built. A TEST knowledge base consists of a highly interconnected network of schemata, each of which is an instance of a schematic prototype.

Failure-mode is the most important schematic prototype. Each failure-mode represents a deviation of the UUT from its standard of correct performance. The failure-modes in a knowledge base are linked in a hierarchy ranging from top-level failure-modes called *concerns* through intermediate-level failure-modes down to leaf-level failure-modes (see Figure 1). Typically intermediate-level failure-modes are functional or subsystem faults, and leaf-level failure-modes are component faults. Failure-modes are interconnected via *due-to* and *always-leads-to* relations. Domain information which affects the behavior of the diagnostic interpreter or the repair strategy is attached as slot values to failure-modes, or other schemata as appropriate.

In addition to failure-modes, the knowledge base contains *datum* schemata which represent evidence-collection actions such as tests, questions and sensor-data acquisition functions. Failure-modes are linked to datums by *has-tests* relations, with meta-information attached to the relation indicating how evidence acquired from the datum(s) is used to confirm/disconfirm the failure-mode. Mechanisms exist to disjunctively and conjunctively combine datums. There are also schemata to represent repairs, documentation, exception conditions, and many other kinds of supporting information.

The TEST problem-solver begins by focusing on an observed or suspected failure-mode. If the failure-mode has occurred or if its status remains unknown, its possible causes are investigated to see if they have occurred. The search process is guided by an underlying representation of the order in which diagnostic experts explore possible causes for identified failure-modes. Heuristic rules can be inserted in the knowledge base to modify search behavior as runtime information is acquired. Diagnosis proceeds until a leaf failure-mode is confirmed, and control is passed to the repair strategy.

3.2. TEST'S repair strategy

TEST queues all confirmed failure-modes that require repair, and by default, repairs them in the reverse order that they are identified. Thus, lower-level, component failures are generally repaired before the higher-level conditions they caused.

Repairs are attached to failure-modes via a *has-repairs* relation. If a failure-mode has links to multiple repairs, they are ordered from most desireable to least according to a domain expert's prior judgement. This ordering incorporates possible tradeoffs between cost and goodness of repair. If the ordering is dependent on runtime information or UUT configuration, this is represented by reorder rules which attach to the *has-repairs* relation and dynamically modify the ordering if their conditions are met.

Once a repair is selected, the repair strategy module leads the technician through the steps of the repair and its subsequent validation. Validation generally consists of checking two failure-modes - the failure which has the repair attached to it and the original concern that triggered the diagnosis - to see whether they still occur.

After performing a repair and validation, the repair strategy selects one of four different sub-strategies, according to an algorithm shown below. TEST follows the selected sub-strategy, modified as necessary to

reflect domain-specific exception information provided in the knowledge base.

```
IF leaf failure-mode has a direct test
  THEN IF leaf failure-mode is still bad
    THEN sub-strategy = BAD REPAIR
    ELSE IF the original concern is fixed
         THEN sub-strategy = SUCCESS
         ELSE sub-strategy = MISDIAGNOSIS
  ELSE
  (no direct test for leaf failure-mode)
    IF original concern is fixed
      THEN sub-strategy = SUCCESS
      ELSE sub-strategy = UNKNOWN PROBLEM
```

The first sub-strategy is for a known bad repair, one which fails to fix the failure-mode directly associated with it. The follow-on diagnostic issues are trivial in this case. TEST knows that regardless of the status of the original concern the leaf failure-mode is still broken. The technician is instructed to check that the replacement parts are good, that they are the proper parts called for by the repair schema, and that they are installed correctly. If the repair still fails, TEST tries to prescribe an alternate repair. If that fails, TEST presumes an error in the knowledge base, either the wrong repair associated with the leaf failure-mode or the wrong method for confirming the failure, and halts.

In the second sub-strategy, success, the repair has fixed the original concern. This is the desired outcome, and causes the repair strategy to halt. Note that the repair strategy does not differentiate between situations where (1) both the leaf failure-mode and the original concern are known to be fixed, and (2) the status of the leaf failure-mode is unknown (eg., not directly testable) but the concern is known to be fixed. This models a heuristic provided by human experts: if the UUT is broken before performing the repair and works afterwards, one can reasonably conclude that the repair fixed all the causes of the original problem.

In the third sub-strategy, misdiagnosis, the repair fixed the leaf failure-mode, but did not fix the original concern. As discussed earlier, this can result from a domain knowledge error, other co-occuring problems, or diagnostic reasoning based on faulty evidence. Since TEST cannot tell if its own knowledge base is in error, it adopts a strategy to discover co-occurring problems if any, and also verify the evidence gained during the first diagnostic pass.

The last sub-strategy, the unknown problem or "broken black box", is the most difficult. A repair has been made, but the original concern is not fixed and there is no direct test for the repaired component. Thus it is impossible to tell if the repair was performed correctly, or to confirm the causal relation between the component repaired and the original concern. Further, the repair may have succeeded but its success masked by additional co-occuring failures. For this case, TEST uses a heuristic from the automotive domain which states *"if you do something to fix a black box and it doesn't work, repeat the repair before trying anything else."* This approach is not entirely satisfying, and clearly has the potential for incurring needless expense in certain cases. But human experts consistently preferred to simply repeat the repair before taking the trouble to go back and verify the diagnosis.

At the completion of any of these four sub-strategies, TEST moves on to repair the next known failure-mode in a similar manner.

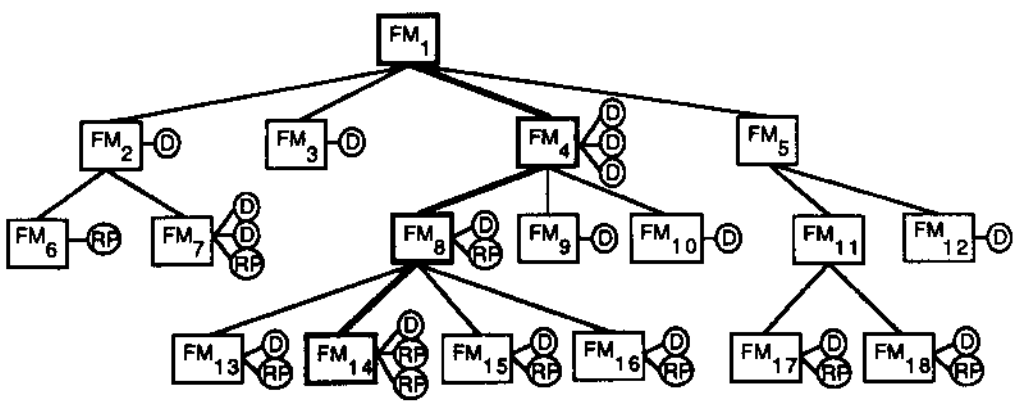

Figure 1: A TEST knowledge base is a hierarchical network of *failure-mode* (FM) schemata, each of which may have *datum* (D) and *repair* (RP) schemata attached via *has-tests* and *has-repairs* relations. Failure-modes are linked "downward" by *due-to* relations, and may also be linked "upward" by *always-leads-to* relations. *Rules* which conditionally modify the knowledge base structure may be attached virtually anywhere, but are not shown here. Diagnosis consists of traversing the failure-mode network, starting at a top-level concern and ending at a leaf node. The final path, shown here in boldface, Is called the *causal chain.*

3.3. Customizing TEST'S repair strategy

The repair strategies as described above may be modified or overridden by specifying exception information in the knowledge base. Exception information is used to modify how a repair's success is verified, how follow-on diagnosis is performed subsequent to repair failure, and the sequencing of multiple repairs.

Verification. As noted earlier, TEST generally verifies repair by checking for the occurrence of two failure-modes: the leaf node directly associated with the repair, and the original concern. But there may be domain-specific reasons to check additional failure-modes, either on the causal chain connecting the two or outside it. Certain repairs may have possible negative side-effects, or certain intermediate failure-modes may have cheap, accurate tests which are more desireable than rechecking the original concern. In other cases, repairs may be trivial and it may be unnecessary to verify them afterwards. TEST provides mechanisms for representing all these exceptions, and for modifying the repair strategy's behavior accordingly.

Alternative diagnosis. If a repair fails, the default strategies described in section 3 can be modified to prune search in domain-specific ways. A domain expert can indicate certain conclusions that can be automatically drawn from specific tests performed as follow-up to a repair (eg, *"if there is no leak in the primary vacuum chamber, and if after replacing the vacuum pump the result of the vacuum test is FAIL, then conclude a leak in the secondary vacuum chamber').* A domain expert can also modify TESTs rather simplistic default behavior of clearing the knowledge base of all evidence after a repair, by specifying which kinds of evidence are likely to be unaffected by certain changes in the UUT's state.

Sequencing. As noted earlier, the default behavior is for TEST to repair failure-modes in the reverse order that they were confirmed. This can be overridden in cases where a repair is needed in order to continue with diagnosis (eg., an empty radiator must be refilled to locate a leak).

4. CONCLUSIONS

We have shown that a diagnostic expert system can be made significantly more powerful by extending its capabilities into the process of repair selection, sequencing and verification. TEST'S repair strategy is a step in that direction. It provides mechanisms in the diagnostic problem-solver for performing default repair strategies, and provides a schematic representation model which permits modification of the repair strategy in domain-specific ways.

TEST succeeds in capturing much of the repair behavior of expert technicians, but it is limited in several ways. Since its knowledge is based on a troubleshooting model, it lacks understanding of the deep structure of the UUT and cannot predict how a repair will affect the reliability of evidence gained prior to that repair, nor can it reason about likely side-effects of repairs. TEST does not currently support temporal representation of evidence, hence it cannot represent or reason about changes to test results or failure-modes over time. And we have not yet fully integrated the repair strategy with TESTs belief maintenance and explanation facilities. As TEST matures with field experience, we expect to correct these problems as required by the user community.

We have only begun to understand the issues of integrating repair strategy with diagnostic reasoning. We are continuing to refine our model of repair behavior, and look forward to the eventual emergence of a sound theory of repair strategy in troubleshooting expert systems.

References

[All 85] All, M., D. A. Scharnhorst. Sensor-based Fault Diagnosis in a Flight Expert System. In *Proceedings of the Second Conference on At Applications.* 1985.

[Bennett 81] Bennett, J.S. DART: An Expert System for Computer Fault Diagnosis. In *Proceedings IJCAI-81.* 1981.

[Bylander 83] Bylander, T., S. Mittal, B. Chandrasekaran. CSRL: A Language for Expert Systems for Diagnosis. In *Proceedings IJCAI-83.* 1983.

[Clancey 84] Clancey, W. Details of the Revised Therapy Algorithm. in Buchanan, B.G., Shortliffe. EH. (editor). *Rule Based Expert Systems.* Addison Wesley, 1984.

[Fink 84] Fink, P.K., J.C Lusth, J.W. Duran. A General Expert System Design for Diagnostic Problem Solving. In *IEEE Workshop on Principles of Knowledge Based Systems.* 1984.

[Hofmann 86] Hofmann, M., J. Caviedes, J. Bourne, G. Beale, A. Brodersen. Building Expert Systems for Repair Domains. *Expert Systems* 3(1):4-11, *January,* 1988.

[Kahn85] Kahn.G. *MUD, a drilling fluids consultant* Technical Report, Dept. of Computer Science, Carnegie-Mellon University. 1985.

[Kahn 87a] Kahn, G., A. Kepner, J. Pepper. TEST, a Model-Driven Application Shell In *Proceedings AAAI-67.* 1987.

[Kahn 87b] Kahn.G. S. From Application Shell to Knowledge Acquisition System. In *Proceedings UCAI-87.* 1987.

[Maletz 85] Maletz, M. C. An Architecture for Consideration of Multiple Faults. In *Proceedings of the Second Conference on AI Applications.* 1985.

[Pepper 86a] Pepper, J. and G. Kahn. Knowledge Craft: an Environment for Rapid Prototyping of Expert Systems. In *Proceedings of the SME Conference on AI for the Automotive Industry.* March. 1986.

[Pepper 86b] Pepper, J., D. Muliins. Artificial Intelligence Applied to Audio Systems Diagnosis. In *Proceedings of the Int'l Congress on Transportation Electronics.* 1986.

(Strandberg 85] Strandberg, C, I. Abramovich, D. Mitchell, K. Prill. Page-1: A Troubleshooting Aid for Nonimpact Page Printing Systems. In *Proceedings of the Second Conference on AI Applications.* 1985.

[Ted68C0 86] Tedesco, L. S. Service Bay Diagnostic System. In *Proceedings of the International Congress on Transportation Electronics.* 1986.