A Data Structure and Algorithm for Fault Diagnosis

Edward L. Bosworth, Jr.
Computer Science Department
The University of Alabama in Huntsville
Huntsville, AL 35899

Abstract

This paper will present results of preliminary research on the design of a Knowledge Based Fault Diagnosis System for use with on-orbit spacecraft such as the Hubble Space Telescope.

This paper discusses a candidate data structure and associated search algorithm from which the Knowledge Based System can evolve. This algorithmic approach will then be examined in view of its inability to diagnose certain common faults. From that critique, a design for the corresponding Knowledge Based System will be developed.

Introduction

The research reported in this paper is focused on the development of an efficient fault diagnosis software system to be used in the operation of on-orbit spacecraft such as the HST (Hubble Space Telescope). There are several factors which indicate the need for an efficient fault diagnosis system. Among these reasons are 1) the desire to detect any fault before it causes damage to the spacecraft (an unlikely but possible event), 2) the desire to have confidence in the accuracy of the scientific data, and 3) the desire to schedule maintenance missions on some sort of cost effective time-line.

The scenario envisioned in this report is an automated fault diagnosis system monitoring the downlinked telemetry reporting on the health status of the spacecraft. This system would act as an intelligent assistant to the operations staff. It would detect abnormalities in the telemetry and deduce the hardware fault causing this indication. It should also be able to project trends in the data and give reasonable predictions of the remaining useful on-orbit time of any replaceable component.

The Data Structure and Associated Algorithm

The efficiency of a computational solution to any interesting problem depends, in general, on the choice of two items: a data structure appropriate to represent the structure of the problem and an algorithm which can operate efficiently on the selected data structure. This observation holds true even in those cases in which the algorithm is complemented by heuristic procedures.

The importance of the data structure arises from the following observation which is generally true: it is usually the case that the more appropriate the data structure the less complex the algorithm and associated heuristics. One criterion used for selecting a data structure is its resemblance to the human representation of the problem being investigated.

One of the primary human tools in fault diagnosis is a design schematic which shows the relations between the functional components of the system. There are two data structures which are analogous to a design schematic: a Directed Acyclic Graph (DAG) and a Tree. Both of these are hierarchical structures which can represent naturally the functional hierarchy found in design schematics. In the data structures this hierarchy is represented by "parent-child links" where a parent node can have several child nodes and represent the fact that a functional component in the design schematic can have a number of functional subcomponents.

Both a DAG and a Tree have at least one designated node which has no parent node. This node is called a "root node". A tree, by definition, has exactly one root node. A DAG may have many such nodes. The concept of a unique root node in the data structure has a strong analogue in the functional design schematic: the entire system. For this reason, the choice of data structures is limited to a Tree or a "Tree-like DAG", the latter being a Directed Acyclic Graph restricted to having one root node.

The main difference between a Tree and a Tree-like DAG is that the nodes of the former are constrained to have exactly one parent node whereas the nodes of the latter may have more than one parent node. This difference will impact the efficiency of representation of the design schematic. If a component is the design schematic can be a part of only one superior component, then the Tree data structure is the preferable representation. If a component may be a part of more than one superior component, then a Tree representation will have to duplicate nodes for that component, attaching a duplicate as a child node to each node which represents one of the superior components. Since the DAG does not require this artificial duplication, it is the data structure selected for this research.

In addition to the Tree-like Directed Acyclic Graph, an auxiliary data structure is used in order to improve the efficiency of the node creation algorithm. This auxiliary data structure is to be used to detect duplicate names and avoid the creation of duplicate nodes in the DAG. In order to use this auxiliary data structure, one must remember that a node in any graph has both an ID and a label. The ID is the variable by which the program accesses the structure representing the node. In LISP this would be called the structures "print name". The label is a name associated with the node. In this data structure, the name of the node is the name in the design schematic of the the component represented by the node. The requirement is to avoid generation of duplicate nodes for the same component in the schematic; i.e., nodes with the same label.

The auxiliary data structure chosen for this is a Hash Table which will store pairs of the form (key, value). The key for an entry will be the node label and the value will be the node ID; thus we have (Node-Label, Node-ID). Any attempt to create a node to represent a named component in the design schematic must first check the Hash Table. If the component be already represented, its Node-ID will be returned. Otherwise a new node may be generated and named as usual.

There is one restriction which must be observed when using an auxiliary data structure. Both the Directed Acyclic Graph and the Hash Table must be defined and accessed as a single abstract data structure. More specifically, there must be a single set of constructor and accessor functions which treat the two data structures and an interdependent pair and enforce the logical relations between the two. Otherwise, the data in the two will become inconsistent and thus useless.

The Directed Acyclic Graph was implemented in VAXLISP by use of the COMMON LISP structure. The following describes a node:

```
(Defstruct (Component
     (:conc-name Node-)
     :predicate)
"A node for representing a component in the design schematic"
  (Name Nil)              ;The name in the design schematic
  (Subcomponents Nil)     ;Note the default values.
  (Contained-In Nil)
  (Search-Seq  0))
```

The Hash Table was implemented as a COMMON LISP global variable.

```
(Defstruct *Component-List* (Make-Hash-Table :Size 197))
```

As mentioned above, the Directed Acyclic Graph and Hash Table must be accessed as a single abstract data type. A typical function is the one which creates a new node. It first checks the Hash Table to avoid making a duplicate. If it continues, it first updates the Hash Table and then creates the node.

```
(Defun Create-Component (Schematic-Name)
 "Creates a node to represent the component with
the specified name in the schematic"

    (Unless (Gethash Schematic-Name *Component-List*)
       (Let ((Node-ID (Gensym "NODE-")))
            (Setf (Gethash Schematic-Name *Component-List*)
             Node-ID)
            (Set Node-ID
               (Make-Component :Name Schematic-Name)) )))
```

Having established the data structure for representing the components, it is now time to discuss the design of an algorithm to do the searching required by fault diagnosis. This design actually has two such algorithms, SEARCH and SWEEP, built around the concept of a search sequence number.

Search sequence numbers are a generalization of the concept of node markings found in many graph and tree search algorithms. Node marks are generally thought of as Boolean variables having the values TRUE or FALSE. An alternate representation of the node mark would be a search sequence having only the permissible values on 0 or 1.

In the search sequence approach, there is a global variable which counts sequentially the searches undertaken during the current session. This variable is passed as an argument to the search procedure. As the procedure visits each node, it processes the node only if the nodes search sequence number is less than the current search sequence. If processed, the node is marked with the current search sequence, is expanded, and its subnodes evaluated for possible search.

The SWEEP procedure is called periodically to reset the search sequence of each node to 0 and to reset the global search sequence variable to 1. The nodes are visited by a simple Depth First Search. One should note that this exhaustive search is called much less frequently than the directed search mentioned above. This results in a reduced overhead due to calling SWEEP.

The algorithm SEARCH is a Best First Search with iterative deepening. It is called with two parameters - a node ID and a search sequence number. At each level, the node is examined to see if it is marked with the current search sequence number. If it be so marked, the next node in the search priority list is examined. Should the node not be so marked, it is given the current search sequence number and examined. Part of the examination is obtaining the subcomponent list and merging that with the rest of the search priority list to form a new search priority list. Nodes which appear more than once will be given a higher search priority on the assumption that if two failed components share a subcomponent then that subcomponent is suspect.

Conclusions and Directions for Future Work

While it seems obvious that an automated fault diagnosis system would be of considerable benefit in the operation of on-orbit spacecraft of the complexity of the Hubble Space Telescope, it is also apparent that an algorithmically based system will not be sufficiently sophisticated.

One flaw in an algorithmically based system is its inability to reason about faults that do not correspond to failed components in the design schematic. A simple example of such a fault is a bridging fault or short circuit, both of which represent components which are not present in the design schematic.

One of the major modifications which will be necessary is the design of a heuristic which can make reasonable modifications to the data structure representing the design schematic in those cases in which the observed fault is not consistent with the normal schematic. This heuristic will include representations of the physical components in order to postulate plausible bridging faults and short circuits. Such systems are discussed extensively in the research literature [1,2,3,4,5].

References

1. Davis, R.; Diagnostic Reasoning Based on Structure and Behavior; Artificial Intelligence 24 (1984) 347 - 410

2. de Kleer, J. and Williams, B.C.; Diagnosing Multiple Faults; Artificial Intelligence 32 (1987) 97-130

3. Genesereth, M.R.; The Use of Design Descriptions in Automated Diagnosis; Artificial Intelligence 24 (1984) 411-436

4. Keravnou, E.T. and Johnson L.; Competent Expert Systems, McGraw Hill, 1986.

5. Reiter, R.; A Theory of Diagnosis from First Principles; Artificial Intelligence 32 (1987) 57-95.