Chapter 5

Inducing Diagnostic Inference Models from Case Data

John W. Sheppard
ARINC Incorporated

Keywords: Diagnostic inference models, case based reasoning, diagnostics, system test, certainty factors, Dempster-Shafer

Abstract: Recent attention to using "case-based" reasoning for intelligent fault diagnosis has led to the development of very large, complex databases of diagnostic cases. The performance of case-based reasoners is dependent upon the size of the case base such that as case bases increase in size, it is usually reasonable to expect accuracy to improve but computational performance to degrade. Given one of these large case bases, it is advantageous to attempt to induce structure from the case base whereby the diagnostic process can be made more efficient. In addition, certainty properties of case based reasoning make fault diagnosis difficult, in which case inducing structure and applying a method for reasoning under uncertainty becomes advantageous. In this chapter, we discuss an approach to analyzing a diagnostic case base and inducing a compact knowledge base using the diagnostic inference model with which efficient diagnostics can be performed. We then apply an approach to reasoning using Dempster-Shafer theory to improve the diagnostics using the resultant model.

1. INTRODUCTION

In the process of developing an optimized approach to system test and diagnosis, it is desirable to apply as much available information to the task as possible. Test and diagnosis is a process of information gathering and analysis, and several approaches have been applied to the problem. Techniques classified as "advanced" frequently fall in the realm of artificial intelligence and can include rule-based methods, model-based methods, and instance-based methods.

Rule-based methods reason with sets of heuristics (i.e., rules of thumb) to draw inferences from test results to make a diagnosis. Sometimes these rules include information on ways to "optimize" the test process. This approach proceeds from the premise that the rules correspond to a reasoning process that represents approaches "experts" apply to problem solving. Model-based methods use a model of the system to be tested to determine "optimal" approaches to testing. Diagnosis involves reasoning about this model under the premise that anything that can be concluded about the model would also be reflected in the actual system. Instance-based methods treat the test problem "extensionally." In other words, rather than capturing rules for inference or modeling the system to be tested, instance-based methods tackle the problem by storing examples of past experience. The premise associated with this approach is that most problem solving involves considering and adapting past experiences rather than using some underlying domain theory or reasoning process.

Diagnosis is a favorite problem for researchers in artificial intelligence and operations research because of the difficulty of the problem, the generality of the problem, and the applicability of a variety of techniques to solving the problem. As one might expect, no single approach can be claimed as the "best" approach to diagnosis. For this reason, it makes sense to consider combining approaches, taking advantage of the best features of each of the individual techniques to yield a robust, "multi-strategy" approach to solving the problem.

In this chapter, we discuss ideas for combining an instance-based method (case based reasoning) with a model-based method (diagnostic inference modeling) to improve the robustness of diagnostic models and diagnostic problem solving. The primary objective of this exploration is to consider an approach to constructing diagnostic inference models from past experience to produce robust models of the system to be tested. The advantages of such a combination include robust knowledge (as reflected in the stored experiences), increased efficiency (resulting from analysis of the generated model), and improved diagnostic accuracy.

2. DIAGNOSIS WITH CASE BASED REASONING

Case based reasoning (CBR) is a method of reasoning that combines elements of instance-based learning and data base query processing. Typically, CBR systems consist of the following elements:

1. A case base
2. A case retrieval system
3. A case adaptation system
4. A case modification system
5. A case update system

These systems work together is what is called the "CBR" cycle. This cycle is illustrated in Figure 1 and consists of four activities–case retrieval, case reuse, case revision, and case retention.

When a problem is presented to the CBR system, it is treated as a "new case." This case is matched with other cases in the case base (also referred to as "memory") until the case most similar to the new case is found. The process of identifying the most similar case is referred to as *retrieval* and may use general knowledge about the problem to guide the search. Once a case is retrieved from the case base, it is reused by the CBR system to recommend a solution to the problem. The process of *reuse* applies general knowledge about the problem to adapt the solution associated with the retrieved case to the new problem. The resulting solution is then recommended to the user. Ultimately, a solution will be found for the problem at hand. This solution may be different from the recommended solution. At this point, the solution associated with the case is further *revised* to account for the actual solution. This revision may include adding, deleting, or modifying information in the case to facilitate proper application of the case in the future. Finally, the new case (with the revised solution) is *retained* in the case base for future application.

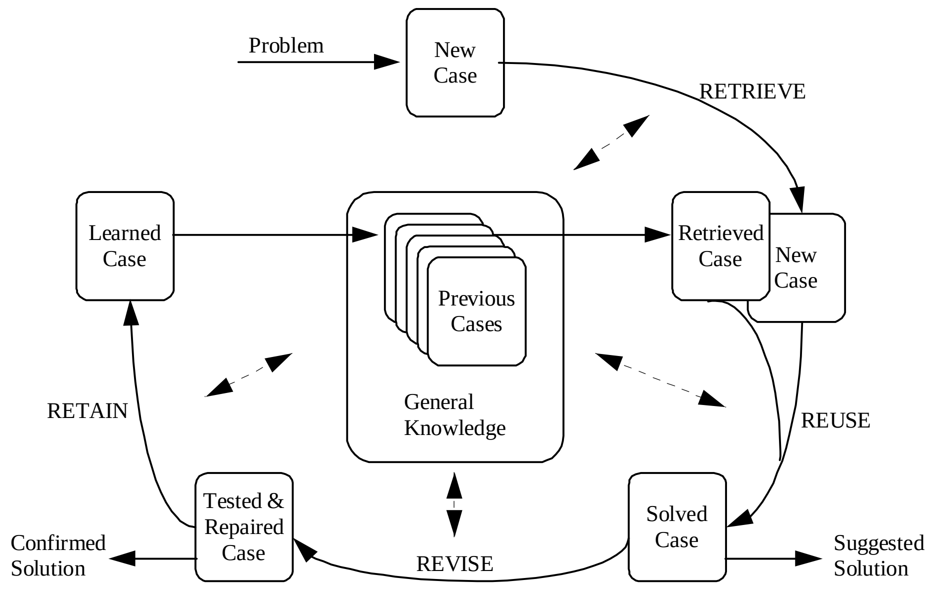


Figure 1. CBR Cycle (Aamodt and Plaza, 1994)

Test and diagnosis can use CBR in several ways. The simplest method involves defining a case as a collection of test results and attempting to determine an appropriate diagnosis given these results. This use of CBR is equivalent to a classification problem as characterized by instance-based learning (IBL). When using the IBL approach, the CBR cycle is simplified in that no adaptation during reuse is required. Further, revision only occurs when the diagnosis was wrong. At that point, only the diagnosis is changed in the case. Finally, the retrieval process is very simple. All of the cases are nothing more that feature vectors with an associated diagnosis $\langle f_1, f_2, \ldots, f_n; d_i \rangle$. The features in the feature vector correspond to test results and may be unknown. Retrieval then consists of "matching" the new case with all of the cases stored in the case base and selecting the most similar case.

When considering possible similarity metrics, numerical features are frequently compared using a member of the family of L_p norms. An L_p norm is defined to be

$$L_p(\mathbf{x}', \mathbf{x}'') = \left(\sum_i (x_i' - x_i'')^p \right)^{1/p}.$$

The most common values for p are 1, 2, and ∞ and yield "Manhattan" distance, Euclidean distance, and max-norm distance respectively. Specifically, these metrics can be computed as

$$L_1(\mathbf{x}', \mathbf{x}'') = \sum_i (x_i' - x_i'')$$
$$L_2(\mathbf{x}', \mathbf{x}'') = \sqrt{\sum_i (x_i' - x_i'')^2}$$
$$L_\infty(\mathbf{x}', \mathbf{x}'') = \sum_i \max\{x_i', x_i''\}$$

If we were using pass/fail results for testing, we would use either L_1 or L_2 (which would be equivalent). Note this is exactly what is done with fault dictionary-based diagnosis. With real values, we would most likely use L_2. Symbolic results are a bit more complicated and would require something like Stanfill and Waltz's "value difference metric" (1986). Regardless of the metric, retrieval would be done as

$$\mathbf{case} = \underset{\forall \mathbf{c} \in \mathbf{CASE\_BASE}}{\arg\min} \{\delta(\mathbf{new}, \mathbf{c})\}.$$

A common variant to this "nearest-neighbor" method is called "*k*-nearest neighbor" and involves retrieving the k nearest cases from the case base and combining (or voting among) the recommended diagnoses. This approach has been demonstrated to approach the Baye's optimal solution as k increases. Other variants involve weighting the solutions based on distance and weighting the features based on significance (or some other factor).

Note that nearest neighbor classification assumes that the mathematical "space" corresponding to the classification problem is a *metric space*. A metric space is defined to be a pair $\langle\Sigma,\delta\rangle$ consisting of a set of points Σ and a distance measure δ. Further, this distance measure is defined to be single-valued, non-negative, and real-valued for all members of the set Σ, i.e., $\forall s_i, s_j \in \Sigma, \delta(s_i, s_j) \in \Re^+ \cup \{0\}$ with the following properties:

1. Reflexivity: $\delta(s_i,s_j) = 0$ *iff* $s_i = s_j$,
2. Symmetry: $\delta(s_i,s_j) = \delta(s_j,s_i)$,
3. Triangle Inequality: $\delta(s_i,s_k) \leq \delta(s_i,s_j) + \delta(s_j,s_k)$.

This assumption will be important when we assess the expected performance of case-based methods in fault diagnosis.

A second approach to applying CBR to diagnosis considers a complete "plan" for diagnosis as a case. This approach assumes the plans can be determined in connection with different situations and that the situations can be characterized prior to diagnosis. The case features would be this characterization, and the case solution would be the recommended plan. Due to the complexities of developing these plans and characterizing the context of diagnosis, this approach is not usually used.

3. FAULT DICTIONARIES AS CASE BASES

One common approach to fault isolation related to case-based reasoning is based on the fault dictionary. Fault dictionaries define a mapping from combinations of input vectors and output vectors to faults. Formally, this is represented as $FD: I \times O \to F$ where *FD* is the fault dictionary, *I* is the space of input vectors, *O* is the space of output vectors, and *F* is the space of faults. At a more basic level, this can be represented as $FD: \{0,1\}^n \times \{0,1\}^m \to F$. This representation makes the fact the vectors are binary explicit. In the simplest case, diagnosis can be performed with a fault dictionary by finding a direct match between the input/output vectors and a fault in the dictionary. Indeed, with a proper model, high confidence tests, and a reasonable fault universe, many faults will be identified in this manner.

For illustration purposes, we will use a simple digital circuit (Abramovici, Breuer, and Friedman, 1990). This circuit is given in Figure 2. From this figure and assuming a single-stuck-at fault model, we can identify 26 possible stuck-at faults. Each stuck-at fault is denoted as x_i where x is a letter matching the line where the fault occurs, and i is either 0 or 1 (denoting stuck-at-0 or stuck-at-1 respectively). We close the fault universe by defining a special "fault" in which no fault has been detected and denote this *nf*. The fault dictionary would then include the input vectors (i.e., the patterns applied to lines *a*, *b*, and *c*) and the expected response vector (in this case the value at *m*). Also associated with that entry would be the list of faults detected should the response be in error.

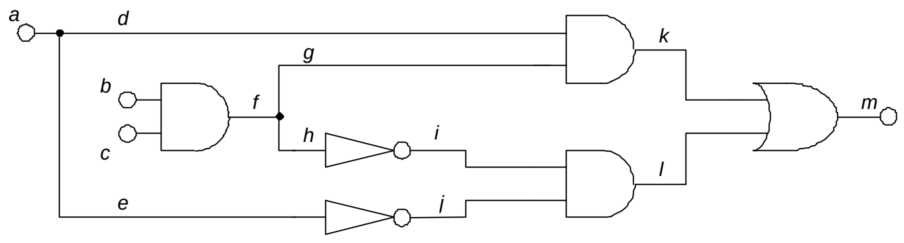


Figure 2. Sample combinational circuit.

In the following sub-sections, we will discuss how to use this information to construct the fault dictionary and how to diagnose faults with the fault dictionary. We will then discuss the problem of diagnosis given inexact matches with elements in the dictionary and describe the nearest neighbor approach for diagnosis with inexact matches.

3.1 Constructing a Fault Dictionary

To explain how to construct a fault dictionary, we will walk through the building of the fault dictionary for the circuit given in Figure 2. In this circuit, we see that there are only three input lines, therefore there are only eight possible input vectors (disregarding timing and faults other than stuck-ats). For our example, we can examine all eight inputs; however, in general, enumerating all possible vectors would be too costly. If the circuit had been a sequential circuit, then the three input lines might require several additional tests because of the sensitivity of the circuit on the previous state of the circuit. Not surprisingly, this combinatorial explosion is significantly worse for larger circuits. Several tools such as LASAR (Richman and Bowden, 1985; Grant, 1986) provide assistance to the modeler in developing input vectors and detecting stuck-ats and other faults at output vectors.

Limiting ourselves to the combinational case (and the example in Figure 2), we begin constructing the fault dictionary by considering the possible input patterns. Each input pattern can be regarded as a test. For example, one test

Table 1. Ambiguity groups for sample circuit.

Number	Ambiguity Group	Number	Ambiguity Group
1	a_0	8	g_1
2	a_1	9	i_0, h_1, l_0, j_0, e_1
3	b_1	10	i_1, h_0
4	c_1	11	j_1, e_0
5	d_1	12	k_0, d_0, g_0
6	f_0, b_0, c_0	13	k_1, l_1, m_1
7	f_1	14	m_0

might be the pattern (0 1 1). Tracing through the circuit, we would expect the output of the circuit to be (0). If the value is (1) then a fault must be present in the circuit. The question then becomes, what failure modes (i.e., stuck-at faults) can cause the erroneous output? Again, examining the circuit identifies a_1, b_0, c_0, d_1, f_0, i_1, h_0, k_1, l_1, or m_1 as possible causes. Similarly with the pattern (0 0 0), we would expect the output of the circuit to be (1) and an output of (0) will implicate one of the following faults: a_1, b_0, b_1, c_0, c_1, d_1, f_0, f_1, h_0, i_1, j_1, k_0, k_1, l_1, or m_1.

Completing the analysis finds several failure modes are “ambiguous,” meaning no test vectors can differentiate them. These ambiguity groups, which were taken from (Abramovici *et al.*, 1990) are shown in Table 1. The approach used for determining ambiguous faults is called “fault collapsing” and consists of identifying lines in the circuit that will have identical values regardless of input or fault because of the logical nature of the gates in the circuit. For example, if we examine the initial AND gate with inputs *b* and *c*, we note that any of b_0, c_0, and f_0 must be indistinguishable because either b_0 or c_0 (or both) will force *f* to have a value of zero, whether or not *f* is faulty. This approach to ambiguity analysis is incomplete, as we will show later, but provides a first cut on the ambiguity in the fault dictionary. From this point forward, we will refer to a particular ambiguity group by the first member of the group.

As mentioned above, the fault dictionary can be constructed in one of two ways. The first approach includes an entry for all of the input/output vectors, and each entry includes the list of faults detected.[1] For the circuit in Figure 2, this

[1] In an attempt to improve the robustness of the standard fault dictionary (based on the single stuck-at fault model), Richman and Bowden (1985) introduced the concept of a “possible detection” in their modern fault dictionary. A possible detection is one that “may” be

means 16 entries would be required; however, we can eliminate all of the "nominal" cases (i.e., the cases where the outputs are correct) since the only associated fault would be *nf*.[2] This reduces the number of entries in the dictionary to eight. For example, entries to the table for the example vectors would be

0 1 1 1 : a_1 d_1 f_0 i_1 k_1.
0 0 0 0 : a_1 b_1 c_1 d_1 f_0 f_1 i_1 j_1 k_0 k_1.

Diagnosis using this form of fault dictionary consists of collecting the sets of detections corresponding to erroneous outputs and taking the intersection of these sets. The fault or faults reported would be given by this intersecting set. For example, suppose in addition to patterns (0 1 1) and (0 0 0), we also ran pattern (1 1 1). The entry in the fault dictionary for this pattern would be

1 1 1 0 : a_0 f_0 k_0 m_0.

Assuming both of the vectors (0 1 1) and (1 1 1) yielded erroneous outputs, we would take the intersection of the two sets which would lead to a diagnosis of f_0 (intersection of the set $\{a_1\ d_1\ f_0\ i_1\ k_1\}$ and $\{a_0\ f_0\ k_0\ m_0\}$). Note that if we had a circuit with more than one output, we could construct separate fault dictionaries for each output, "sensitizing" the appropriate paths leading to that output to determine the faults detected. These dictionaries could be combined into one large dictionary, but this may lead to further confusion should there be a mismatch between the target signature and the signatures in the dictionary.

The second approach to representing the fault dictionary is to construct a table in which each input vector corresponds to a row in the table. The columns of the table correspond to all of the ambiguity groups in the circuit. Each cell in the table contains the expected output from the circuit. The fault dictionary for our example circuit would be represented in this form as in Table 2. This dictionary assumes eight tests as follows:

t_1 : 0 1 1 t_2 : 1 1 0 t_3 : 1 0 1 t_4 : 1 1 1

detected when an unexpected test result is encountered. Possible detections arise when indeterminate states propagate through the system.

[2] Unless there are undetected failure modes in the circuit. Of course, in this case, these nondetections would be ambiguous with *nf*, and we decided to refer to the ambiguity group by a representative member, namely *nf*. Thus the reduction is still valid.

Table 2. Fault dictionary for sample circuit.

	a_0	a_1	b_1	c_1	d_1	f_0	f_1	g_1	i_0	i_1	j_1	k_0	k_1	m_0	nf
t_1	0	1	0	0	1	1	0	0	0	1	0	0	1	0	0
t_2	1	0	0	1	0	0	1	1	0	0	1	0	1	0	0
t_3	1	0	1	0	0	0	1	1	0	0	1	0	1	0	0
t_4	0	1	1	1	1	0	1	1	1	1	1	0	1	0	1
t_5	1	0	0	1	1	1	0	1	0	1	1	1	1	0	1
t_6	1	0	1	1	1	1	0	1	0	1	1	1	1	0	1
t_7	1	0	1	0	1	1	0	1	0	1	1	1	1	0	1
t_8	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0

t_5 : 0 0 1 t_6 : 0 0 0 t_7 : 0 1 0 t_8 : 1 0 0

Diagnosis using the second format matches the results of running the tests with the columns in the table. For example, suppose we run all eight tests and get (1 0 0 1 1 1 1 0) as the set of responses. This pattern would match both d_1 and i_1, indicating ambiguity between the two associated groups.[3] The significant observation to be made here is that ambiguity is determined by the actual tests used to test the circuit, and selecting a subset of possible test vectors could result in a variety of different ambiguity groups. For example, if we only evaluated t_1, t_2, t_3, and t_4, we would find a_1 is now ambiguous with both d_1 and i_1.

3.2 Diagnosis with Nearest Neighbor Classification

The most common approach to generating and using fault dictionaries for diagnosis is based on the second approach discussed in the previous section. We will show later in this chapter how the first approach can be used to generate more robust diagnostics. For now, we will consider the approach of matching vectors of test results. In particular, we want to consider the case where the vector of test results fails to match any of the columns in the fault dictionary.

Given the single stuck-at fault model, we assume the circuit simulation accurately reflects the performance of the actual circuit. In other words, we

[3] The previous set of ambiguities was constructed assuming all of the test vectors were available. This analysis was performed using a fault-collapsing technique (Abramovici *et al.* 1990). However, as we see, d_1 and i_1 are also inherently ambiguous, as are g_1 and j_1, illustrating that fault collapsing is not sufficient for robust ambiguity analysis.

assume the only faults of interest to us are stuck-at faults, these faults are accurately represented in the circuit model, and only one of these faults will be encountered at a time. Given these assumptions and the fact digital circuit models are deterministic (i.e., the outputs are directly determined by the inputs and, in the case of sequential circuits, the internal state), whenever the fault signature fails to match any exemplars in the fault dictionary, the circuit must be exhibiting behavior that was not represented in the circuit model (i.e., the problem lies in the fault model, not the test results).

Debaney and Unkle (1995) assert, “In practice, it is very seldom that an observed fault signature has an exact match in the fault dictionary.” This result motivates the need for an “inexact” pattern matching algorithm when using a fault dictionary. Inexact matches may be caused either by submitting noisy inputs to the circuit, introducing noise in the output, omitting an important signature from the dictionary, or having to deal with multiple faults or indeterminate states, thus invalidating the signatures. Noisy input data may yield unexpected results, but since we have control over the inputs, noise at this level indicates a problem in the test equipment or the test itself. Noisy output data indicates either a problem in the test equipment, a failure mode not included in the model, or an error in the circuit model. It is possible that vectors have not been included in the fault dictionary which, when applied to the circuit, would yield unexpected results, but having control over the test process should preclude applying those input vectors.

The current practice for processing inexact matches in fault dictionaries applies various distance measures to find the column in the dictionary that most closely matches the target vector. The literature reports two different distance measures used with the fault dictionary:

1. Hamming distance: $\delta(s_i, s_j) = \sum_b \left| s_i^b - s_j^b \right|$, where *b* denotes the bit compared,
2. Overlap metric: $\delta(s_i, s_j) = \dfrac{\left| s_i \cap s_j \right|}{\left| s_i \cup s_j \right|}$.

Given these distance measures, the fault dictionary can be characterized as a metric space. However, we still need to determined whether any distance measure satisfying the above properties is reasonable to apply to our problem.

As mentioned above, the presence of possible detections complicates the matching algorithm in that matches are considered but mismatches are ignored. For example, the LASAR approach uses a variation on nearest-neighbor matching in which it is assumed all of the vectors initially match the target vector, and scores for signatures in the fault dictionary are penalized when a mismatch occurs. With mismatches on definite detections and non-detections, a large penalty is applied, but when the mismatch occurs on a possible detection, only a small penalty is applied. This has the effect of "weighting" the test attribute based on the certainty of the test being able to detect (or clear) a particular fault.

4. FAULT DICTIONARY COMPRESSION

Generating and using full fault dictionaries or case bases for complex systems, in general, is not practical due to the intense computational requirements, even when limited to simplified fault models. For this reason, several approaches exist to reduce the size of the case base by either not generating entries that would add little diagnostic information or by compressing the case base following generation (Tulloss, 1978; Tulloss, 1980; Ryan, 1994).

The idea behind most of the compression techniques is to apply a greedy heuristic to decide whether or not to include a case in the case base. The greedy approach is used since the problem of determining the smallest set of examples to isolate all faults is reducible to the set-covering problem which is known to be *NP*-complete, In general, a heuristic evaluation function (HEF) is used (e.g., number of new faults detected or isolated), and cases are added that improve the value of the HEF the "most." Alternative approaches begin with all of the cases or signatures in the case base and "drop" faults with HEFs with the lowest value.

These compression techniques are analogous to the problem of selecting relevant features in pattern recognition. The two most common approaches to feature selection are step-wise forward selection and step-wise backward selection (Devijver and Kittler, 1982). In step-wise forward selection, features are added until classification accuracy begins to degrade, and the features are selected incrementally based on maximizing classification accuracy. In step-wise backward selection, features are removed using the same criteria. These approaches are analogous to determining which tests to include in the fault dictionary and form a method of compressing the dictionary along the test axis.

In the pattern recognition literature (Dasarathy, 1991), approaches to reducing the size of the instance base that more closely relate to the common fault dictionary compression methods are known as editing methods. Frequently, it has been found that editing can improve performance of nearest neighbor classification, especially when the instance base has a large number of noisy features or noisy examples. Early work by Wilson (1972) showed that examples could be removed from a set used for classification and suggested that simply editing would frequently improve classification accuracy (in the same way that pruning improves decision trees (Mingers, 1989)). Wilson's algorithm classifies each example in a data set with its own *k* nearest neighbors. Those points that are incorrectly classified are deleted from the instance base, the idea being that such points probably represent noise. Tomek (1976) modified this approach by taking a sample (> 1) of the data and classifying the sample with the remaining examples. Editing then proceeds using Wilson's approach. These approaches are analogous to determining which fault signatures to include in the fault dictionary and form methods of compressing the dictionary along the fault axis. Any compression method discards information. In the most benign case, it discards "unneeded" information resulting in no effective loss in information. However, when faced with the possibility of being unable to match signatures exactly, it is unclear what information is unneeded, and compression may aggravate the matching problem.

5. PROBLEMS WITH INSTANCE BASED DIAGNOSIS

For maintenance to be cost effective, the troubleshooting strategy applied to a unit under test must be efficient and effective. Efficiency is important for reducing the cost of doing maintenance by optimizing the required maintenance resources. Effectiveness is important since ineffective maintenance leads to increased logistics costs and sparing requirements. Effective includes the important attribute of accuracy. Ineffective repair and its effects can be attributed directly to lack of effective troubleshooting.

We claim that applying the nearest neighbor classification method to outcome-based diagnosis such as that used with most case-based reasoners and fault dictionaries leads to ineffective diagnostics and, thereby, ineffective repair. In fact, we find in (Abramovici *et al.*, 1990) that the following problems are already known to exist in using fault dictionaries. First, the computational requirements for computing fault dictionaries is quite high, making generating a dictionary for a large circuit with long test sequences expensive. Thus dictionary compression is required. Second, fault dictionaries depend on a pre-defined "fault universe" and "test universe." In other words, the fault dictionary will find only the faults specified in the dictionary, and these faults can be found only with the specified set of tests. The primary assumption here is that the fault is a member of the set defined in the fault dictionary and the output vector of tests is in error in one or more bits. Nearest neighbor would treat this like a noisy signal problem, finding the existing candidate with the closest match of attributes. Finally, Abramovici *et al.* (1990) note that the nearest-neighbor approach to matching inexact patterns in the dictionary, while effective in many cases, "is not guaranteed to produce the correct diagnosis."

In the following sections, we will focus on this third issue and illustrate that, in fact, nearest neighbor is a poor choice for handling inexact matches in the general case. We will discuss sources of error in the nearest neighbor approach and suggest that, due to the discrete nature of the problem, nearest neighbor is less appropriate than other available approaches for diagnosis with the same data. Further, we will show that the primary cause of nearest neighbor's poor performance is its focus on the classification (i.e., fault) space, rather than the attribute (i.e., test) space.

5.1 Sources of Error Using Nearest Neighbor

One of the simplest ways to understand how nearest neighbor classification works is by casting the approach in a geometric framework. In general, we can think of points stored in a data base as representing concepts to be identified. When presented with a new point, we look at the "exemplars" stored in the data base to help us decide which concept best classifies the new point. In a sense, we are looking for a dividing line between concepts in the data base and look for the side of the line on which the new point falls.

In fact, this is exactly how nearest neighbor works. Consider the points shown in Figure 3. If the points represent columns in the fault dictionary, diagnosis consists of finding the point (called an "exemplar") in the data base that most closely matches the point to be classified (i.e., the test results). Geometrically, the "dividing line" between two exemplars used to determine the nearest neighbor is a line perpendicular to the line connecting the two exemplars.

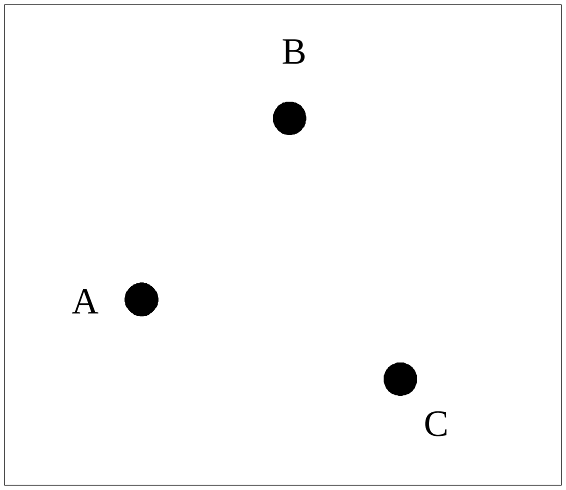


Figure 3. Nearest neighbor exemplars.

Further this line intersects the connecting line at the midpoint between the two exemplars. This is shown with the same three points in Figure 4. The set of dividing lines between the exemplars (i.e., the dividing hyperplanes in higher dimensions than two) is called a *Voronoi diagram*. Nearest neighbor classification consists of determining in which region the point to be classified falls and assigning the corresponding label.

Assuming the data stored in the dictionary is correct,[4] at least three situations arise in which nearest neighbor classification may result in error. The first case arises when the attributes (e.g., the test results) are irrelevant or insignificant. This situation is illustrated in Figure 5. In this figure, assume only one attribute (corresponding to the *x*-axis) is significant for classifying the points as either "black" or "white." Next assume an irrelevant attribute has been added to each of the points (corresponding to the *y*-axis). For all practical purposes, we can assume the second attribute can be modeled as white noise. The vertical line in the figure represents the correct decision boundary between "black" and "white," but the jagged line represents the Voronoi diagram corresponding to the exemplar set. If a point to be classified falls inside any of the regions labeled

[4] We will not consider here the case where erroneous data may have been introduced into the exemplar set. Nevertheless, it should be clear that the ability to classify is limited by the quality of the exemplars.

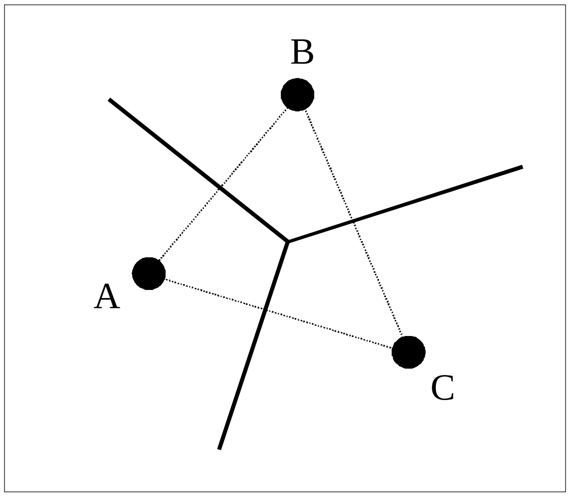


Figure 4. Voronoi diagram for example points.

“error,” that point will be mis-classified. In the case base, an irrelevant attribute is analogous to considering an output other than in the system under test. Since we have complete control over which observable outputs to consider, this source of error should not be a problem; however, care should be taken not to include tests that add nothing to diagnostic resolution. Such tests introduce noise since they are “irrelevant.”

The second source of error arises when a significant attribute is missing from the exemplar set. For example, suppose the points in Figure 6 are correctly classified along the decision boundary shown. This boundary assumes both attributes (corresponding to both the *x*- and *y*-axes) are present. But suppose the fault simulator fails to consider the *y*-axis. In other words, a particular test vector is not processed by the simulator. This is equivalent to projecting all of the points into the *x*-axis. In Figure 6, we see the decision boundary then overlaps which leads to another source of error. In the case base, the overlap is equivalent to increasing ambiguity between failure modes, and can arise as a direct result of case base compression when reducing the set of tests.

The third source of error arises from the fact that nearest neighbor can be considered a method for function approximation. In the case where the decision boundaries do not consist of linear segments, the actual boundary can only be approximated by nearest neighbor. In outcome-based testing, all of the stored

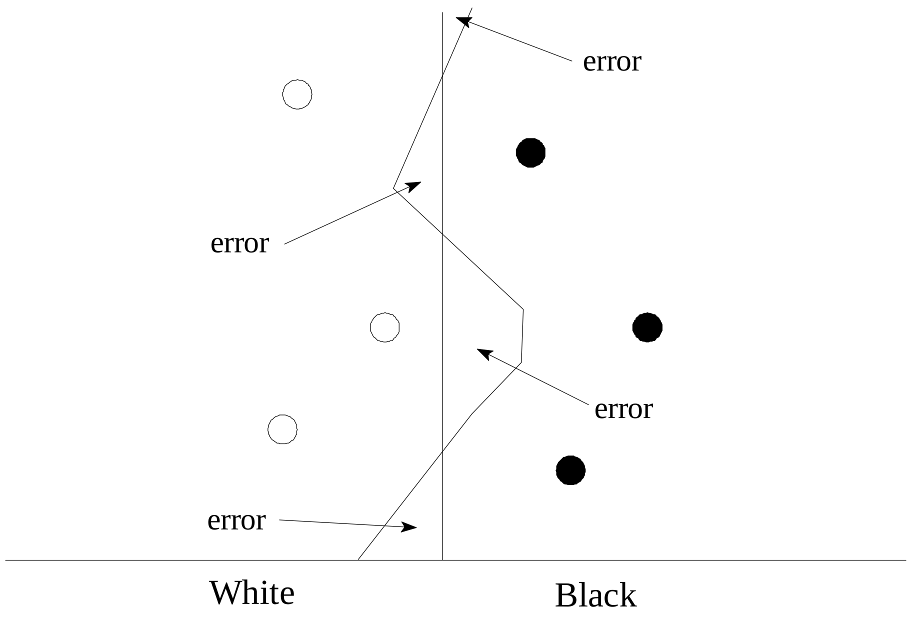


Figure 5. Errors arising from irrelevant attributes.

points exist on the corners of an *n*-dimensional hypercube, so nearest neighbor is able to model the decision boundaries exactly. However, if points are missing from the space of possible points, then it is possible that the decision boundaries will not be modeled correctly. For example, suppose the white point shown in Figure 7 is missing. Then presentation of that point for classification later will result in an error since all points on the lower left side of the decision boundary will be classified as "black." Unfortunately, this is analogous to what happens when compressing the set of fault signatures, except in such a case, the missing point corresponds to a missing failure mode. Thus classification is impossible since a class is completely unrepresented.

5.2 The Appropriateness of Nearest Neighbor

In determining the appropriateness of nearest neighbor for outcome-based diagnosis, we must examine the characteristics of diagnosis and outcome-based

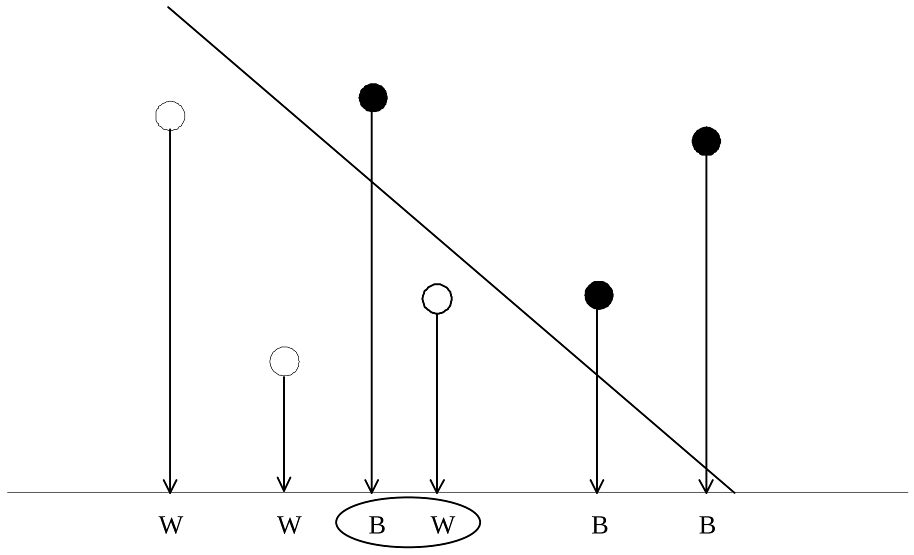


Figure 6. Error arising from missing attributes.

case bases. Then we consider the characteristics of the exemplars to be used for diagnosis and look at the impact of these characteristics on the potential for error. Given our discussion on error sources, in this section, we will only consider the case where we may have missing attributes, noisy attributes, or missing exemplars.

When we construct a case base, we assume several characteristics of the diagnostic process. First, we assume we will be able to apply all of the stimuli (i.e., all the tests are available and performable). Further, we assume that the results we obtain from testing reflect the circuit under test (whether failed or not). Finally, we assume that all of the failure modes of interest to us are modeled in the fault represented in the case base.

Nearest neighbor classification is appropriate when the exemplars in the data base are *representative* of possible examples to be classified. The exemplars are intended to model the underlying distributions of the classification space. Cover and Hart (1967) motivated using the nearest neighbor rule with the following:

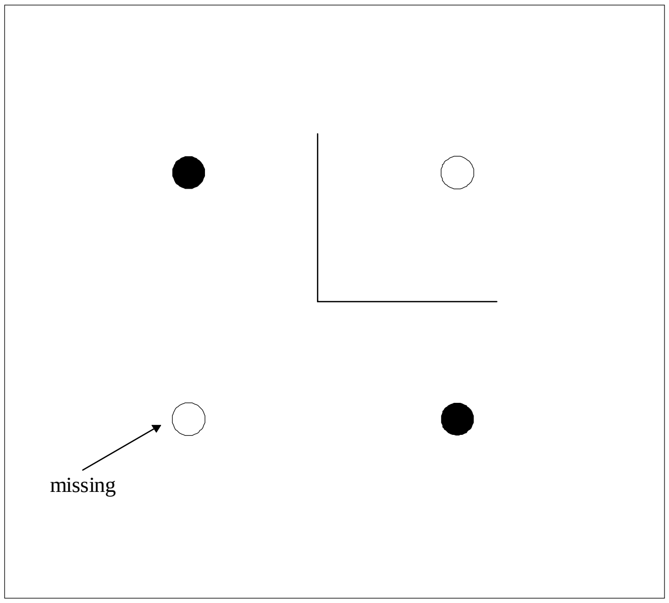


Figure 7. Errors from missing data.

> If it is assumed that the classified samples (x_i, θ_i) are independently identically distributed according to the distribution of (x, θ), certain heuristic arguments may be made about good decision procedures. For example, it is reasonable to assume that observations which are close together (in some appropriate metric) will have the same classification, or at least will have almost the same posterior probability distributions on their respective classifications.

Unfortunately, the set of exemplars defined by outcome-based testing are not independently identically distributed (i.e., representative of random variables associated with the underlying classes or failure modes) according to the distribution of possible points. This is because most case bases are constructed to maximize detection, and the concern is only to provide at least one input vector to detect each failure mode. Further, faults within the system affect more than one test showing some form of dependence.

Other possible reasons outcome-based diagnostics are not well suited for nearest neighbor include the following. For the exemplar set to be effective, it must be representative of the underlying probability distributions. This assumes sufficient sample size for each of the classes (i.e., failure modes) in the case base. Unfortunately, computational complexity precludes generating and using a comprehensive case base. Many tools apply a "detect limit," *n*, which results in signatures corresponding to some fault being discarded when other signatures detect that fault at least *n* times. This approach forces under-representation of the classification space which violates the assumption of the points representing the underlying distribution. Also when the underlying distributions of the attributes and classes are not adequately sampled and those distributions are discrete, it is nearly impossible for the nearest neighbor classification procedure to be reliable.

In addition, the exemplars generally are not independent; although, an assumption of independence is necessary for the single stuck-at fault model to apply. Independence breaks down when we consider the effects of one fault on other parts of the system (e.g., it is not uncommon for the presence of a fault to cause another fault). Thus it would appear nearest neighbor is not an appropriate decision rule for outcome-based diagnostics in the general case.

5.3 Nearest Neighbor Diagnosis and the Error Radius

Under very specific conditions, it may be that nearest neighbor will provide good diagnosis when using outcome-based testing. Recent work in network fault management and alarm correlation is analogous to fault-dictionary-based diagnosis (Kliger, Yemini, Yemini, Ohsie, and Stolfo, 1995). In this work, the relationships between test results and failure modes are represented using coding theory where a "codebook" is constructed to include the set of alarms pertinent to diagnosis and the specific failure modes of interest. Diagnosis then consists of applying a nearest neighbor classification rule to the codebook with a set of symptoms to determine the associated failure mode.

In the network fault management problem, the possibility of error in the test results is quite high; therefore, handling erroneous test results in diagnosis is critical. To guarantee correct diagnosis in the presence of noise, Yemini *et al.* (1994) examine the characteristics of the codebook and define precise conditions under which nearest neighbor classification is appropriate.

Central to nearest neighbor being applicable to diagnosis with the case base (or a codebook) is understanding the nature of an error radius. The *error radius* is defined to be the minimal Hamming distance among the code vectors in the codebook. The codebook is constructed in a way similar to constructing a fault dictionary. In particular, each row of the codebook corresponds to a failure mode, and each column corresponds to a symptom. These symptoms are the same as the tests in the fault dictionary. A cell in the codebook receives the value ‘1’ when the associated symptom appears in the presence of the failure mode and ‘0’ otherwise. Thus the fault dictionary can be converted into the codebook by converting the actual outputs of the circuit to ‘1’ or ‘0’ depending on whether the outputs are expected to be in error.

This model can be referred to as a “causality likelihood model.” An entry in the matrix can be interpreted as the likelihood that the test will detect the presence of the associated failure mode. In the deterministic case where we assume the model is correct, the likelihood values would either be ‘0’ or ‘1’. In the more general case, we can associate a value between zero and one corresponding to the probability of the test failing given the presence of the failure mode.

When considering the possibility of error in the fault signature, we wish to match the signature to the codebook recognizing we will not be able to obtain an exact match. In such a case, we would like to know the number of errors in the signature we can tolerate and still find the correct fault. A measure of the tolerance to error is given by the error radius. For example, if the error radius is one, this means two signatures in the codebook differ by exactly one bit. If that bit is in error, we will not be able to distinguish between the two failure modes. From this we can conclude that the higher the error radius, the more resilient the codebook is to noise.

Several important assumptions underlie the coding method and the use of the error radius. The key assumption is that, as with the fault dictionary, the coding method assumes the underlying model is correct and complete. Correctness requires the code vectors to be accurately modeled and that error results in the testing process only. Completeness requires that all relevant failure modes are included and fully represented by the code vectors; errors arising from an unanticipated failure mode are assumed not to occur or not to be relevant to the diagnosis underway.

If in fact the cause of an erroneous fault signature is the test process or the propagation of the signal through the circuit and not the presence of an

Table 3. Codebook for sample circuit.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
a_0	0	1	1	1	0	0	0	1
a_1	1	0	0	0	1	1	1	0
b_1	0	0	1	0	1	0	0	0
c_1	0	1	0	0	0	0	1	0
d_1	1	0	0	0	0	0	0	0
f_0	1	0	0	1	0	0	0	0
f_1	0	1	1	0	1	1	1	1
g_1	0	1	1	0	0	0	0	1
i_0	0	0	0	0	1	1	1	0
i_1	1	0	0	0	0	0	0	0
j_1	0	1	1	0	0	0	0	1
k_0	0	0	0	1	0	0	0	0
k_1	1	1	1	0	0	0	0	1
m_0	0	0	0	1	1	1	1	0
nf	0	0	0	0	0	0	0	0

unanticipated fault in the circuit, then testing within the error radius may be effective. Table 3 shows a codebook corresponding to the circuit in Figure 2 and based on the reduced fault dictionary in Table 2. Given this codebook, we find the error radius is one (if we collapse all ambiguity into a single row in the matrix). Also, of the $2^8 = 256$ possible code vectors, only 13 are represented. Thus this simple circuit will not be able to process noisy fault signatures well at all.

In larger circuits, the number of possible code vectors increases exponentially, yet the number of signatures included in the codebook does not (since it is not generally practical to generate all possible vectors). Unless ambiguity increases, the subset of signatures must result in an error radius less than or equal to the error radius of the complete set. To prove this, note that eliminating a test vector is equivalent to eliminating an attribute in the exemplar set (not a whole exemplar). Comparing exemplars in which the attribute value is different, deletion of the attribute must decrease the Hamming distance, thus potentially decreasing the error radius. Comparing the exemplars in which the attribute value is the same, deleting the attribute has no effect on the Hamming distance, thus not changing the error radius. Thus the potential for error cannot decrease with the smaller attribute set.

6. DIAGNOSIS WITH DIAGNOSTIC INFERENCE MODELS

In Chapter 6, we discuss using a diagnostic inference model (also called an information flow model) to capture relationships between test outcomes and diagnoses (Sheppard and Simpson, 1998). The concept of the diagnostic inference model is closely related to the fault dictionary, except that it abstracts the concepts of a test and a fault to address testing at the "system level" (Sheppard and Simpson, 1991; Simpson and Sheppard, 1994).

Inherently, diagnostic inference models are difficult to construct. This difficulty arises from the fact that DIMs require the definition of a comprehensive set of tests, diagnoses (usually based on known failure modes) and relationships between the two. Further, special inference types (e.g., asymmetric inference or cross-linked inference) may be required to capture accurately the inferences derivable from the test results. Unfortunately, such difficulty increases the potential for inefficient or even incorrect diagnosis. Further, as systems increase in complexity, the likelihood of erroneous models increases. Two questions naturally follow from this problem: 1) How does one develop models that minimize the chance of an error? 2) If errors occur, how does one identify and correct them?

Results from machine learning research suggest potential answers to both questions. The most common approach applied is to apply simulation or fault insertion to generate examples that capture failed behavior and determine test-to-fault relationships. This is probably the most reliable approach to learning or constructing a model; however, it tends to take an inordinate amount of time before useful models are constructed. Related to this approach is using historical data (when available) to construct the model. The next section discusses an approach to do just that.

7. CONSTRUCTING A DIAGNOSTIC INFERENCE MODEL FROM CASE DATA

The primary goal of this section is to discuss methods for constructing diagnostic inference models from case data used in a CBR system. A straightforward approach to doing this, but one that is not likely to generalize well is to treat each case as a unique class (i.e., diagnosis). Then the signature for that class would correspond to the feature values associated with the case. For multi-value features, the binary equivalents in the IFM would be a pairing of the feature label with the feature value. For example, suppose we have a case base with the following three cases.

Case	**Test 1**	**Test 2**	**Test 3**	**Test 4**
c1	Pass	Fail	Unknown	Unknown
c1	Unknown	Fail	Fail-Lo	Unknown
c2	Fail	Unknown	Fail-Hi	Pass

Note that two of the cases yield the same diagnosis, and one of the tests has three possible outcomes (assuming all tests can pass). Since the IFM assumes a test dependency indicates detection of a fault, an indication of a test passing just means that the test does not depend on the associated fault. Unknown outcomes were not evaluated but may have been inferred.

If we treat each case as a separate class, or diagnosis, then this set of cases would result in the following model:

Case	**Test 1**	**Test 2**	**Test 3-Lo**	**Test 3-Hi**	**Test 4**
c1-a		X			
c1-b		X	X		
c2	X			X	

The problem with this approach is that diagnosis would occur (with high confidence) only when a signature is matched exactly (i.e., when one of the cases is experienced again). But this means there is no facility for generalization. This is not a desirable feature, so we would like to provide a method for "combining" cases with the same diagnosis to yield greater generality.

Consider the same three cases. We still need to map multiple-outcome tests into a set of several binary tests, so the columns would remain the same. We only have one case for c2; therefore, we would retain this case in the model as in the previous example. But we should be able to combine the two cases for c1 into a single signature in the model. To do this, we need to ensure that the two cases do not conflict. If they do conflict, then we are back to creating separate signatures.

For this example, the cases do not conflict. This is clear because all of the tests whose values are known for all of the cases agree. If they disagreed, then there would be a conflict. Since there is no conflict, the simplest process for building the signature is to take the union of the signatures from the first model. Doing this yields,

Case	Test 1	Test 2	Test 3-Lo	Test 3-Hi	Test 4
c1		X	X		
c2	X			X	

The problems arise when there are conflicts in the test results. Previously, we provided an intuitive "definition" of conflict as occurring when two cases with the same diagnosis have attributes with different values. Since the case base is not supposed to hold any incorrect cases (i.e., cases where the either the diagnosis or the attribute values were in error), we should be safe in assuming that these conflicts arise only when the attribute can legally take on *either* value for that diagnosis.

Consider the following example. In some sense, it is the simplest example in that it reveals a direct conflict for the same diagnosis. This conflict is on Test 1.

Case	Test 1	Test 2	Test 3	Test 4
c1	Pass	Fail	Unknown	Unknown
c1	Fail	Unknown	Pass	Unknown

For the other three tests, we can simply take the union as we did before. For Test 1, since it can either pass or fail, we need to either treat these as two different conclusions (as we did in the naive example above), or we need to consider what could lead to the conflict. Assuming the tests are correct and reliable (which is a standard assumption when constructing IFMs), we can assume Test 1 is actually *asymmetric*. In particular, we note that Test 1 detected c1 in the second case, but when Test 1 passed, we were not able to rule out c1. This would yield a single signature of

Case	Test 1 (–)	Test 1 (+)	Test 2	Test 3	Test 4
c1	X		X		

A brute-force approach for handling this type of conflict when constructing the IFM is to declare all tests in the model to be *fully asymmetric*. This means that the positive and negative inference lists may be different. Next, group the cases such that all cases with the same diagnosis are in the same group. For a particular group, define a signature in the IFM. Consider all of the failed tests first. If a case exists in the group with a failed test, enter the diagnosis as a dependency of that test on both the positive and negative inference side. Next consider all of the passed tests. If a case exists in the group with a passed test where a dependency has been entered, remove that dependency on the positive inference side. Do not worry about tests whose values are unknown—they will be treated as not depending on the fault until evidence to the contrary is encountered. Finally, after the model is constructed, the inference lists can be compared. If any test has the same list for both the positive and negative inference sides, that test can be converted back into a symmetric test.

Now consider the case where we have a multiple-outcome test (e.g., Test 3 with possible values of pass, fail-hi, and fail-lo). This may yield a "conflict" such as the following:

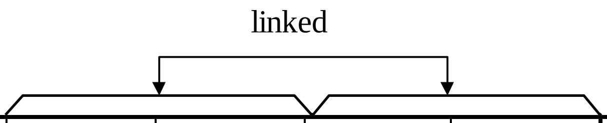
Case	**Test 1**	**Test 2**	**Test 3**	**Test 4**
c1	Pass	Unknown	Fail-Lo	Unknown
c1	Unknown	Fail	Fail-Hi	Unknown

As before, this situation indicates the test can take on two legal values in the presence of this fault. The difference, however, is that the same "test" succeeds in detecting the fault. If we construct the model as before, we will still yield non-conflicting signatures. Thus,

Case	**Test 1**	**Test 2**	**Test 3-Lo**	**Test 3-Hi**	**Test 4**
c1-a		X	X		
c1-b		X		X	

would be an acceptable model. Note that the test failure for Test 2 is entered into both signatures since it is not a point of conflict. In this case, it is also desirable to create a *linked outcome* between Test 3-Lo and Test 3-Hi such that if either fails, the other must pass. This is because it is impossible for Test 3 to have the value of Fail-Lo and Fail-Hi at the same time. But this leads to a pass/fail conflict similar to above.

To combine these two signatures, we would have to treat each "side" of Test 3 as asymmetric as we did before. Thus, Test 3-Lo and Test 3-Hi would both be fully asymmetric tests, but c1 would be absent from the dependency list of the positive inference sides of both tests. Then we would have a single signature.



Case	**Test 1**	**Test 2**	**Test 3-Lo (–)**	**Test 3-Lo (+)**	**Test 3-Hi (–)**	**Test 3-Lo (+)**	**Test 4**
c1		X	X		X		

For cases where the structure of the case is more complex than a simple "feature vector," the structure would need to be flattened into the feature-vector form to facilitate mapping into the IFM. This should be straightforward since most case representations that are not flat use the structure to provide means of speeding up search and comparison. Leaf values in the structure (possibly with some propagation downward through the case structure) should be sufficient for constructing the feature vectors.

It should be apparent that the completeness of the model that results from mapping the case data depends heavily on the richness of the cases stored in the case base. Areas of the case base that do not adequately represent the corresponding set of faults may lead to situations in which the model-based approach leads to inaccurate or imprecise results. This is not unexpected. In fact, similar problems are likely to result when applying the CBR system should test results lead to that part of the case base. Recalling the observation by Cover and Hart, we see why it is important to have a representative sample of case data for performing diagnosis with CBR or for inducing an IFM from the case data.

8. REASONING UNDER UNCERTAINTY WITH DIMS

By now it should be apparent that great care should be used when diagnosing faults with an outcome-based approach. Because of the inherent difficulties in processing erroneous test data in these approaches, we developed two alternative approaches to processing this data which are presented in Chapter 6 (Sheppard and Simpson, 1998). The alternatives consider test results as evidence for or against the presence of a fault in the system. Test results are processed sequentially, and the evidence supporting or denying the presence of a failure mode is attributed to the set of failure modes in the system. These approaches are based on the Dempster-Shafer method for reasoning under uncertainty (Dempter, 1968; Shafer, 1976) and certainty factors as incorporated in the *MYCIN* system (Shortliffe, 1976).

In related research, Denœux (1995) attempted to overcome some of the drawbacks of the nearest neighbor classification rule and proposed a modification to *k*-nearest neighbor using Dempster-Shafer. Denœux notes that "the main drawback of the voting *k*-NN rule is that it implicitly assumes the *k* nearest neighbors of a data point *x* to be contained in a region of relatively small volume, so that sufficiently good resolution in the estimates of the different conditional densities can be obtained." He goes on to point out that in practice, "the distance between *x* and one of its closest neighbors is not always negligible, and can even become very large outside the regions of high density." Since an outcome-based case base defines points on the corners of an *n*-dimensional hypercube, it is reasonable to assume this "pathological" condition holds.

The proposed solution is to gather "evidence" from the neighbors and use this evidence to classify the point *x*. The basic method Denœux uses is to identify the *k* nearest neighbors and combine the evidence provided by these neighbors for each of the classes under consideration. Specifically, he compute the "basic probability assignments" (BPA) as $\mu^{s,(i,j)} = \mu^{s,i} \oplus \mu^{s,j}$ where *s* denotes the sample to be classified, and x^i and x^j are two points which are in the set of nearest neighbors Φ^s that belong to the same class C_q. The BPAs are computed using sets of nearest neighbors of each class, Φ_q^s, and the associated evidence is combined for each class. Using this formulation, the rankings provided by both support and plausibility are consistent, so the class assigned to x^s is simply the class with the maximum support (or plausibility).

While providing improvement over the standard *k*-nearest neighbor rule, all of the problems described previously still apply. While this approach tends to reduce the reliance of diagnosis on homogeneous regions of points in the instance base, it still assumes neighboring regions provide information about the point to be classified. This is unlikely to hold in outcome-based diagnostics, and we believe the evidence to be provided for classification (i.e., diagnosis) comes from *test* information rather than neighboring *diagnoses*. This approach is fundamental to the diagnostic inference modeling approach.

9. DIAGNOSIS OF A SIMPLE CIRCUIT

In this section, we will walk through an example diagnosis with the circuit in Figure 2 and illustrate how an inference method such as the Dempster-Shafer approach can still provide a reasonable answer in the presence of uncertainty, even when nearest neighbor classification does not. First we will consider the performance of nearest neighbor on the sample circuit, and then we will apply Dempster-Shafer inference with the diagnostic inference model on some example diagnoses. For this example, we must assume we have received an erroneous fault signature. We will need to define confidence values for the test results and process the test results through the model.[5]

9.1 Diagnosis with Nearest Neighbor

First, we will consider the ability of nearest neighbor to process erroneous fault signatures in a fault dictionary. For the following demonstration, we considered only the sample circuit and used both Hamming distance and the overlap metric. The matching procedure was limited to 1-NN, and we expect worse results for k-NN with $k > 1$ since faults are only represented by one signature each. For each fault, we considered all possible fault signatures that can be generated with one through eight bits in error. We then compared the results of using 1-NN with the expected fault and recorded the number of correct diagnoses. The results of these experiments are given in Table 4.

[5] Recall our algorithm for applying Dempster-Shafer includes calculation for support of an "unanticipated result." This special conclusion is included to address issues related to conflicting test results. Fortunately, the presence of conflict does not affect the relative positions of the conclusions in the table since support for the unanticipated result only modifies the normalizer for other support values.

Table 4. Accuracy using nearest neighbor on a fault dictionary.

Bit Errors	1	2	3	4	5	6	7	8
Hamming Dist.	82	110	42	9	0	0	0	0
correct diagnosis	79%	30%	6%	1%	0%	0%	0%	0%
incorrect diagnosis	21%	70%	94%	99%	100%	100%	100%	100%
Overlap	76	91	57	19	0	0	0	0
correct diagnosis	73%	25%	8%	2%	0%	0%	0%	0%
incorrect diagnosis	27%	75%	92%	98%	100%	100%	100%	100%
Total Cases	104	364	728	910	728	364	104	13

From this table, we see some characteristics of introducing error into the fault signature to be matched. First, we see that the higher the number of bits in error, the lower the accuracy in matching, down to a limit of 0% accuracy. Second, the performance of Hamming distance compared to the overlap metric is very close. In fact, we conjecture that the differences are not statistically significant; although, we do not have sufficient data to perform a significance test. Third, the lowest error rate (i.e., one bit error) yielded very poor performance on this circuit (between 21% and 27% error). This should not be a surprise given the previous discussion on the appropriateness of nearest neighbor, but it may be disconcerting to those who apply nearest neighbor in their diagnostic systems.

Given the poor performance on nearest neighbor, we will take two cases in which nearest neighbor fails to find the correct failure mode and process those cases with the diagnostic inference model and the Dempster-Shafer methodology. We will then reconstruct Table 4 using Dempster-Shafer. The first case will use a fault signature with one bit in error, and the second case will use a signature with two bits in error. Given the rapid degradation after two bits (since more than 25% of the bits are now wrong), we will assume Dempster-Shafer will be affected similarly with the high error percentages.

In selecting our two test cases, we want nearest neighbor to fail to find the correct fault. Since we know the number of incorrect bits, we will identify two failure modes in the fault dictionary whose Hamming distance corresponds to twice the number of incorrect bits minus one and ensure the proper number of differentiating bits are in error. This will result in selecting the wrong failure mode, but with a signature that is not in the fault dictionary.

9.2 One-Bit Error with Dempster-Shafer

For the first case, we note that the Hamming distance between c_1 and *nf* is two. In other words, two tests detect the presence of c_1, and if both of those tests are in error, no fault will be detected. This is a common occurrence in testing, and effective diagnostics as well as effective testing are necessary to combat this problem. For our example, we will assume one of the two tests capable of detecting c_1 fails to make the detection (i.e., the test passes). Without loss of generality, we will select t_2 to pass when it should fail. We will also assume sufficient test data has been gathered to identify t_2 as a problem test, and we will reduce its confidence to 0.75. All other test confidences will be set to 0.99

The results of processing all eight test vectors through the Dempster-Shafer calculations with the diagnostic inference model are given in Table 5. The normalized values for evidential probability, though quite low, show the leading candidates for diagnosis are c_1 and *nf*. This result is consistent with what we would expect given the nature of the test data.

Another interesting result when considering the test data applied to the diagnostic inference model came when we omitted t_2 from the calculations. Although both c_1 and *nf* were still the leading candidates, the differences in probabilities between c_1 and *nf* were such that c_1 could be declared with greater confidence to be the fault. In fact, this makes sense given the new test result from t_2 contradicts c_1 as the hypothesis.

Given this result, we decided to perform a sensitivity analysis on the erroneous test to determine at what point the incorrect diagnosis would be returned. For this analysis, we varied the confidence in the test outcome from 0.01 to 0.99 and observed the relative difference in probability of the correct answer (i.e., c_1) and the nearest neighbor (i.e., *No Fault*). We expected the choice from Dempster-Shafer to flip at some point and return the incorrect answer. As we can see from Figure 8, however, this did not happen. In fact, we found that Dempster-Shafer returned the correct example regardless of the confidence value; however, as confidence increased, the difference between c_1 and *No Fault* converged to the point they were no longer significantly different. In fact, the hypothesis generation routine returned both faults as possible answers.

Table 5. Dempster-Shafer calculations with one bit error.

Failure Mode	Support	Plausibility	Probability
a_0	0.043	0.411	0.0505
a_1	0.079	0.629	0.0767
b_1	0.067	0.629	0.0768
c_1	**0.111**	**0.906**	**0.1057**
f_0	0.070	0.629	0.0760
f_1	0.052	0.411	0.0514
i_0	0.093	0.753	0.0902
i_1	0.084	0.753	0.0895
j_1	0.056	0.535	0.0652
k_0	0.084	0.753	0.0895
k_1	0.043	0.411	0.0505
m_0	0.079	0.629	0.0767
nf	**0.098**	**0.876**	**0.1024**

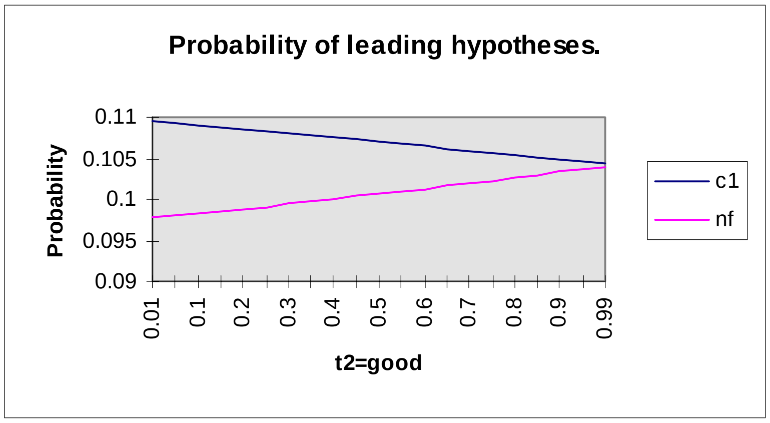


Figure 8. Sensitivity of confidence on one-bit error.

9.3 Two-Bit Error with Dempster-Shafer

For the second test case, we identified two failure modes whose Hamming distance was three. This indicated only three test results distinguished the two conclusions. Given the application of nearest neighbor with two tests

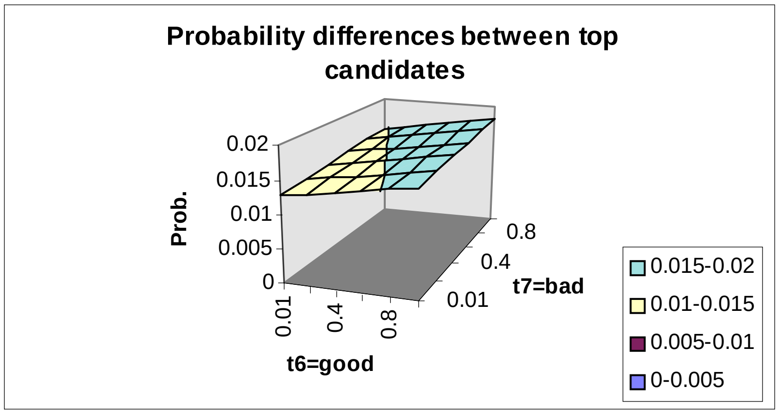


Figure 9. Sensitivity of confidence on two-bit error.

results in error when the two tests were among the three distinguishing the conclusions, we would expect the wrong conclusion to be identified.

Assume the failure mode present in the system is a_1. Tests t_5, t_6, and t_7 differentiate this failure mode from failure mode i_1. Without loss of generality, suppose both t_6 and t_7 are in error. Again, we assume we have sufficient test data to warrant assigning confidence values of 0.75 to these two tests and 0.99 for all other tests. The results of processing all eight test vectors through the Dempster-Shafer calculations with the diagnostic inference model are given in Table 6. The normalized values for evidential probability, this time, show the leading candidates for diagnosis are a_1 and i_0. Again, this result is consistent with what we would expect given the nature of the test data since the Hamming distance between a_1 and i_0 is only 1.0. The erroneous conclusion drawn by nearest neighbor of i_1, though third, never appears as a member of the hypothesis set.

As before, we examined the results of the Dempster-Shafer calculations without the “conflicting” test results from t_6 and t_7. No conflict was evident without these test results. Further, we once again found that the hypothesis without these two tests was a_1 by itself. It was only in the presence of the conflicting test results that i_1 was added to the hypothesis, but a_1 remained the preferred failure mode. Also, when only t_6 was included, though conflict was now present in the test results, the conflict was not sufficient to add another failure mode to the hypothesis; a_1 remained the sole failure mode in the hypothesis set.

Table 6. Dempster-Shafer calculations with two bit errors.

Failure Mode	Support	Plausibility	Probability
a_0	0.035	0.258	0.0392
$\boldsymbol{a_1}$	**0.131**	**0.813**	**0.1180**
b_1	0.066	0.535	0.0790
c_1	0.029	0.318	0.0465
f_0	0.073	0.535	0.0798
f_1	0.093	0.535	0.0820
$\boldsymbol{i_0}$	**0.100**	**0.689**	**0.1012**
i_1	0.087	0.659	0.0966
j_1	0.049	0.381	0.0574
k_0	0.043	0.411	0.0606
k_1	0.080	0.505	0.0768
m_0	0.087	0.565	0.0850
nf	0.056	0.535	0.0779

We also conducted a sensitivity analysis on the confidence values of the two erroneous tests. This time, we varied the confidence values from 0.01 to 0.99 in a factorial study. The results of this analysis are given in Figure 9 and represent the difference in probability between a_1 and i_0. As before the correct fault is always identified as the first choice using Dempster-Shafer, but this time, as confidence increases, the difference does not become as small.

9.4 Dempster-Shafer and Nearest Neighbor Compared

To further compare the differences between the Dempster-Shafer approach and nearest-neighbor classification, we computed the accuracy for all bit-error combinations using Dempster-Shafer as we did for nearest neighbor. These results are shown in Table 7. In interpreting this table and Table 4, we can consider the bit errors as corresponding to some amount of lost information. For example, in the two-bit error case, we assume 25% information loss. From this we can see that even one bit error is significant in that it corresponds to 12.5% information loss.

Consider the rows labeled "Correct = 1st." These rows correspond to the analysis when we consider the conclusion assigned the highest probability of being correct. This is analogous to the nearest-neighbor case in which we select

Table 7. Accuracy using Dempster-Shafer on a fault dictionary.

Bit Errors	1	2	3	4	5	6	7	8
Correct = 1st	89	174	150	62	0	0	0	0
correct diag.	86%	48%	21%	7%	0%	0%	0%	0%
incorrect diag.	14%	52%	79%	93%	100%	100%	100%	100%
Correct = 1st$\vee$2nd	104	331	336	66	0	0	0	0
correct diag.	100%	91%	46%	7%	0%	0%	0%	0%
incorrect diag.	0%	9%	54%	93%	100%	100%	100%	100%
Total Cases	104	364	728	910	728	364	104	13

the fault whose signature is closest to the test signature as the most likely diagnosis. Comparing these rows with Table 4, we find that Dempster-Shafer strongly outperforms both Hamming distance- and Overlap metric-based nearest neighbor. In fact, we see that with 37.5% information loss, nearest neighbor performs randomly (i.e., if we randomly select from the 13 possible failure modes meaning any failure mode might be selected with probability 7.7%, we will be correct approximately the same number of times as nearest neighbor with three bits in error). On the other hand, Dempster-Shafer does not reduce to "random" performance until we have 50% information loss. When information loss exceeds 50%, both techniques fail to find the correct diagnosis, and this is not unexpected.

An interesting result with Dempster-Shafer involves examining the number of times the correct answer is either the first or second most likely conclusion identified (shown in the rows labeled "Correct = 1st$\vee$2nd"). Here we find the correct fault a very high percentage of the time, indicating an alternative answer in the event repair based on the first choice is ineffective. In fact, in all cases where the answer was ranked either first or second, Dempster-Shafer still considered it to be a member of the hypothesis set.

A closer examination of the results using Dempster-Shafer yield some interesting observations. If we limit our consideration to the one-bit error case, we find that Dempster-Shafer returns the correct diagnosis as either the first or second choice in all cases. Examining the tests that are in error, we find an interesting pattern. In all cases where the correct diagnosis is second, either t_1 or t_4 is in error, and in five out of the eight cases, both of these tests in error result in the correct diagnosis being listed second. For the other three cases, one of the tests result in the "wrong" answer, but the other does not.

In all cases, we can explain the cause of the error by examining the Hamming distance between the reported most likely diagnosis and the correct diagnosis. In these cases, the Hamming distance is one, and the bit that is different is the bit whose error leads to the wrong diagnosis. This supports the conclusion of the significance of the error radius and seems to indicate that, as long as the number of bits in error is less than the error radius of the signatures in the fault dictionary, Dempster-Shafer will yield the correct result.

We can extend this result by examining the relative Hamming distances between all of the faults in the fault dictionary. Those faults with a large amount of similarity will be expected to lead to incorrect diagnosis in the presence of higher bit errors in the signature. In fact, this is exactly what we observed. With both the two- and three-bit error cases, we found faults i_0, i_1, j_1, and k_0 had difficulty, but the number of other signatures similar to these four signatures (i.e., with Hamming distance less than or equal to three) was high. This further confirms the significance of the information provided by the tests and its ability to distinguish faults in contrast to focusing on the conclusion landscape to determine proper diagnoses.

10. CONCLUSION

In this chapter, we described an approach to generating diagnostic inference models from case data stored in a CBR system. We also provided an approach to reasoning under uncertainty using the resultant model. The rationale for developing this approach was two-fold. First, we wanted to improve the diagnostic process by providing a more compact representation of the diagnostic knowledge. Second, we wanted to provide a mechanism whereby understanding the diagnostic knowledge was possible. CBR systems are typically very slow where IFM-based systems are fast. Further, it is difficult to develop any understanding of the structure and meaning of knowledge implicit in the cases of a CBR system where model-based systems emphasize understanding this structure. Third, given degraded accuracy resulting from case-based diagnosis applied to outcome-based testing, we wanted to provide a means for improving accuracy using the derived models.

The primary disadvantage to any model-based system (including IFM-based systems) is the difficulty in developing the model. While, conceptually, the elements required for IFMs are easy to understand, the process of collecting and synthesizing the data into the IFM is highly labor-intensive and error-prone. It is difficult to dispute the accuracy of case data since it corresponds to actual diagnostic experience (unless erroneous case information is stored). Therefore, the advantage to processing the case data to generate an IFM is that is provides a solid foundation of experience from which to derive the models. The resulting system then enjoys the advantages of both case-based systems and model-based systems while simultaneously minimizing the effects of the disadvantages.

11. ACKNOWLEDGMENTS

The work presented in this paper combines research in diagnostic and machine learning. Consequently, it reflects the advice and counsel of several people. Thanks go to Jerry Graham, Don Gartner, Steven Salzberg, David Aha, Harry Dill, and Warren Debaney. Without their insight and advice, this work would not have been completed.

12. REFERENCES

Aamodt, A. and E. Plaza. 1994. “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches,” *AI Communications*, Vol. 7, No. 1, pp. 39–59.

Abramovici, M., M. A. Breuer, and A. D. Friedman. 1990. *Digital Systems Testing and Testable Design*, New York: Computer Science Press.

Cover, T. M., and P. E. Hart. 1967. “Nearest Neighbor Classification,” *IEEE Trans. on Information Theory*, Vol. IT-13, pp. 21–27.

Dasarathy, B. V. (ed.). 1991. *Nearest-Neighbor Norms: NN Pattern Classification Techniques*, Los Alamitos, California: IEEE Computer Society Press.

Debaney, W. H. and C. R. Unkle. 1995. “Using Dependency Analysis to Predict Fault Dictionary Effectiveness,” *AUTOTESTCON ‘95 Conference Record*, New York: IEEE Press.

Denœux, T. 1995. “A *k*-Nearest Neighbor Classification Rule Based on Dempster-Shafer Theory,” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-25, No. 5, pp. 804–813.

Devijver, P. A. and J. Kittler. 1982. *Pattern Recognition: A Statistical Approach*, Englewood Cliffs, New Jersey: Prentice-Hall.

Dempster, A. P.. 1968. “A Generalization of Bayesian Inference,” *Journal of the Royal Statistical Society*, Series B, pp. 205–247.

Grant, F. 1986. “Noninvasive Diagnostic Technique Attacks MIL-SPEC Problems,” *Electronics Test*, Miller-Freeman Publications.

Kliger, S., S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. 1995. “A Coding Approach to Event Correlation,” *Fourth International Symposium on Integrated Network Management*, Santa Barbara, California.

Mingers, J. 1989. “An Empirical Comparison of Pruning Methods for Decision Tree Induction,” *Machine Learning* Vol. 4, No. 2, pp. 227–243.

Quinlan, R. 1986. “Induction of Decision Trees,” *Machine Learning*, Vol. 1, pp. 81–106.

Richman, J. and K. R. Bowden. 1985. “The Modern Fault Dictionary,” *Proceedings of the International Test Conference*,” Los Alamitos, California: IEEE Computer Society Press.

Ryan, P. 1994. *Compressed and Dynamic Fault Dictionaries for Fault Isolation*, Ph.D. Thesis, University of Illinois at Urbana-Champaign.

Shafer, G. 1976. *A Mathematical Theory of Evidence*, Princeton, New Jersey: Princeton University Press.

Sheppard, J. W. 1996. “Maintaining Diagnostic Truth with Information Flow Models,” *AUTOTESTCON ’96 Conference Record*, New York: IEEE Press.

Sheppard, J. W. and W. R. Simpson. 1991. “A Mathematical Model for Integrated Diagnostics,” *IEEE Design and Test of Computers*, Vol. 8, No. 4, Los Alamitos, California: IEEE Computer Society Press, pp. 25–38.

Sheppard, J. W. and W. R. Simpson. 1998. “Managing Conflict in System Diagnosis,” *System Test and Diagnosis: Research Perspectives and Case Studies*, eds. J. W. Sheppard and W. R. Simpson, Norwell, Massachusetts: Kluwer Academic Publishers, 1998.

Shortliffe, E. H. 1976. *Computer Based Medical Consultations: MYCIN*, Elseview, New York.

Simpson W. R. and J. W. Sheppard. 1994. *System Test and Diagnosis*, Norwell, Massachusetts: Kluwer Academic Publishers.

Stanfill, C. and D. Waltz. 1986. “Toward Memory-Based Reasoning,” *Communications of the ACM*, Vol. 29, No. 12, pp. 1213–1228.

Tomek, I. 1976. “An Experiment with the Edited Nearest Neighbor Rule.” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-6, No. 6, pp. 448–452.

Tulloss, R. E. 1978. “Size Optimization in Fault Dictionaries,” *Proceedings of the Semiconductor Test Conference*, Los Alamitos, California: IEEE Computer Society Press, pp. 264–265.

Tulloss, R. E. 1980. :Fault Dictionary Compression: Recognizing when a Fault may be Unambiguously Represented by a Single Failure Detection,” *Proceedings of the International Test Conference*, Los Alamitos, California: IEEE Computer Society Press, pp. 368–370.

Wilson., D 1972. “Asymptotic Properties of Nearest Neighbor Rules Using Edited Data,” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-2, No. 3, pp. 408–421.