

Creating Small Fault Dictionaries

Brian Chess*, *Member, IEEE* and Tracy Larrabee†, *Senior Member, IEEE*

Abstract

Diagnostic fault simulation can generate enormous amounts of data. The techniques used to manage this data can have significant effect on the outcome of the fault diagnosis procedure. We first demonstrate that if information is removed from a fault dictionary, its ability to diagnose unmodeled faults may be severely curtailed even if dictionary quality metrics remain unaffected; we therefore focus on methods for producing small, lossless dictionaries.

We present a new dictionary organization based on *error sets*, which is amenable to standard data compression techniques. We compare several dictionary organizations and the effect of standard data compression techniques on each of them. An appropriate organization and encoding makes dictionary-based diagnosis practical for very large circuits.

1 Introduction

Integrated circuit manufacturers are constantly trying to decrease the number of faulty parts they produce. By analyzing parts that fail production tests and determining each part's cause of failure, a manufacturer may be able to improve the circuit design or the manufacturing process. Determining the cause of failure for a circuit is known as *failure analysis*.

Fault diagnosis is part of Failure analysis. The objective of fault diagnosis is to form a hypothesis that predicts the physical location of the failure on the chip. A failure analysis engineer uses fault diagnosis results to guide the search for the defect. If the diagnosis is imprecise, the engineer may have an overwhelming physical area to examine. Worse yet, if the diagnosis is inaccurate and the engineer removes the upper layers of the faulty circuit, the actual defect may be destroyed and there will be no way to determine the cause of failure. It is important that a diagnosis be both precise and accurate.

Fault diagnosis begins with the selection of a *fault model*, which describes the behavior of the faults that will be used to explain a faulty circuit's behavior. The popular *single stuck-at* fault model assumes that the circuit contains a wire that has lost its ability to change values. The wire is *stuck-at 1* or *stuck-at 0*. The single stuck-at fault model is widely used because of its simplicity.

The majority of spot defects in modern CMOS technologies cause changes in the circuit description that result in shorts [10], which implies that many defects create *bridging* faults. The bridging fault model assumes that the outputs of two gates have been accidentally joined or bridged. Modeling the logic behavior of bridging faults is an option, but this approach is complicated by the influence of relative drive strengths, bridge resistance, variable logic thresholds, and potential feedback loops [9]. The importance of these factors is compounded by the fact that accurate modeling of a fault's behavior is much more important for diagnosis than it is for production testing: Fortuitous detection is a boon during test, but any unmodeled behavior can result in imprecision and inaccuracy during diagnosis.

*When this work was performed, Brian Chess was with the Department of Computer Engineering, UCSC. His current address is Hewlett-Packard Company, 1501 Page Mill Road MS-6UJ, Palo Alto CA 94304

†Department of Computer Engineering, University of California, Santa Cruz 95064. This work was supported by the Semiconductor Research Corporation under Contract 93-DJ-315 and the National Science Foundation under grant MIP-9011254.

Regardless of the fault models and detection methods used, a simple diagnosis procedure takes place as follows: Each simulated *fault signature* (the response of the circuit in the presence of the fault) is compared to the observed response of the faulty part. If the observed faulty response is identical to a fault signature, the corresponding fault is added to the diagnosis. In practice, more sophisticated techniques are often used to pick a set of signatures that best match the observed faulty response [13].

If a set of faults share the same signature, it is impossible to distinguish between them without additional information. These faults are said to form an *equivalence class* under the applied test set. The best test set for diagnostic purposes will differentiate between all faults that are distinguishable. The *diagnostic expectation* for a set of faults and a set of tests is the average equivalence class size over all faults [17]. For a given set of faults and a set of tests, diagnostic expectation is often used as a metric to judge the expected precision of a diagnosis. The smaller the diagnostic expectation, the higher the precision of the diagnosis is expected to be.

Once a set of tests and a set of faults have been selected, the faults can be simulated against the tests and the resulting signatures stored in a *fault dictionary*. A dictionary-based approach removes the onus of fault simulation from the diagnosis procedure, but if the fault signatures cannot be stored in a space-efficient manner, the resulting volume of data may be overwhelming.

To cope with the volume of data, every dictionary storage scheme consists of a *dictionary organization* and a *dictionary encoding*. Previous researchers have proposed organizations and encodings at the same time, but the decisions are separable. An appropriate organization and encoding makes dictionary-based diagnosis practical for very large circuits. Smaller dictionaries allow for the inclusion of more information, and more information leads to better results.

The dictionary organization is the order and content of the information in the dictionary. When the organization removes information, the resulting dictionary is said to be *compacted*. Dictionary compaction schemes sometimes attempt to remove information from the dictionary without altering the sizes of the equivalence classes the dictionary contains; if a compaction technique that maintains equivalence classes is applied to a fault dictionary, the dictionary is equally useful with and without compaction—with respect to the faults in the dictionary. But if the circuit to be diagnosed contains a fault that is not in the dictionary, information important to the diagnosis of the fault may be lost.

Dictionary encoding encompasses all issues of symbol representation including bit-packing or data compression. The choice of organization affects which encoding methods are most effective, and the most effective encoding methods may motivate changes in organization. We are interested in exploring the effectiveness of standard data compression algorithms as encoders for a variety of dictionary organizations.

When a fault dictionary is used as part of a fault diagnosis procedure, each of the entries in the dictionary must be compared to the observed behavior. The comparison can be accomplished by changing the representation of the observed behavior to match that of the dictionary entries or by changing the dictionary entries to match the form of the observed behavior: the correct choice depends on the format of the dictionary. Most previously published approaches have implied that the observed behavior will be converted into the format of a dictionary entry, but the use of standard data compression techniques for dictionary encoding require that entries be decompressed for comparison: the format of each entry will be changed to match the format of the observed behavior.

Section 2 reviews common dictionary organizations, encodings, and attendant compaction techniques. Section 3 documents an experiment in which a variety of dictionary types are applied to the problem of diagnosing unmodeled faults. The experiment shows that common metrics for evaluating dictionary quality are inadequate for determining a dictionary’s ability to diagnose faults it does not contain—by removing information from the dictionary, compaction can significantly reduce the effectiveness of a fault dictionary even if it maintains the dictionary’s equivalence classes. Section 4 introduces a new dictionary organization based on *error sets* that is amenable to standard data compression techniques. Section 6 compares several dictionary organizations and the effect of standard data compression techniques on each of them.

Fault #	Fault Name
1	primary input w3gat stuck at 1
2	input pin 1 of nand2 1 stuck at 1
3	output pin of nand2 1 stuck at 0
4	output pin of nand2 2 stuck at 1
5	input pin 2 of nand2 4 stuck at 1
6	output pin of nand2 4 stuck at 0
7	output pin of nand2 5 stuck at 0
8	primary input w2gat stuck at 1

Table 1: The example fault list.

Test	Inputs	Outputs
1	11011	11
2	10000	00
3	11000	11
4	10011	01

Table 2: The example test set.

2 Dictionary organizations and encodings

This section reviews several fault dictionary organizations, many of which were described by Ryan, Fuchs, and Pomeranz [17], and then discusses possible encodings of those organizations. Figure 1 gives a small circuit (the C17) to serve as an ongoing example. To keep the example small, only the eight faults listed in Table 1 and the four tests in Table 2 will be considered.

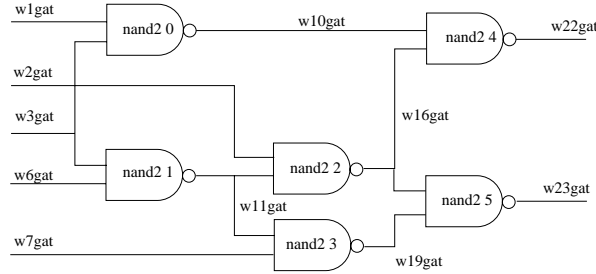


Figure 1: The C17: the example circuit.

Fault #	Test			
	1	2	3	4
1	10	10	11	10
2	00	00	11	00
3	00	00	00	00
4	01	00	00	01
5	01	00	01	01
6	01	00	01	01
7	10	00	10	00
8	11	11	11	11

Table 3: The full response dictionary.

Fault #	Test			
	1	2	3	4
1	1	1	0	1
2	1	0	0	1
3	1	0	1	1
4	1	0	1	0
5	1	0	1	0
6	1	0	1	0
7	1	0	1	1
8	0	1	0	1

Table 4: The pass-fail dictionary.

The *full response dictionary*, which stores the values of all circuit outputs in the presence of each fault for each test, appears in Table 3. This dictionary has the advantage of providing all the information available for a given test set. Further, it can easily be encoded in an efficient bit-packed manner. The number of bits required to store a full dictionary is $F \cdot V \cdot O$ where F is the number of faults, V is the number of vectors, and O is the number of primary circuit outputs¹.

¹If circuit outputs can take on more than two values, more than one bit will be needed to represent each circuit output. This will increase the size of the dictionary by a factor of $\lceil \lg |A| \rceil$ where A is alphabet of possible output values. The remainder of this section assumes that $A = \{0, 1\}$.

Fault #	Test							
	1		2		3		4	
1	1	1	1	0	1	1	1	
2	1	0	0	0	1	1	1	
3	1	0	0	1	0	1	1	
4	1	0	0	1	0	0	0	
5	1	0	0	1	1	0	0	
6	1	0	0	1	1	0	0	
7	1	1	0	1	0	1	1	
8	0	1	1	0	1	1	1	

Table 5: The Compact dictionary.

A *pass-fail* dictionary (Table 4) only stores one bit of information per fault-vector pair—a 1 if the circuit will fail the test in the presence of the fault and a 0 if it will pass. The diagnostic expectation of the pass-fail dictionary is worse than the diagnostic expectation for the full dictionary, but it can be encoded in only $F \cdot V$ bits. If the circuit has a large number of primary outputs, the pass-fail dictionary is significantly smaller than the full response dictionary.

Pomeranz and Reddy’s *Compact* dictionary compaction method uses a greedy algorithm to choose columns from the full response dictionary to augment a pass-fail dictionary [15]. The result is a dictionary with no erosion of diagnostic expectation (for the modeled faults). The Compact dictionary in Table 5 is made up of the pass-fail dictionary and two columns from the full dictionary: Output 1 for Vector 1 and Output 2 for Vector 3. It contains the same equivalence classes as the full dictionary.

Another popular dictionary organization is the *detection* dictionary, shown in Table 6. This dictionary stores only the output patterns with errors for each fault. Since it is unlikely that all faults will be detected by all tests, fewer output sequences need to be stored in the detection dictionary than in the full response dictionary. In its full form, the detection dictionary contains all of the information contained in the full response dictionary, but the detection dictionary is easy to compact—a drop-on-K dictionary is a detection dictionary where the number of detections for each fault is limited to K. One variation on drop-on-K is a *drop-when-distinguished* dictionary, which for a given fault, stops recording vector information as soon as the fault’s equivalence class is of size 1, that is, when the fault is fully distinguished. Very similar to a detection dictionary, but containing the information of a pass-fail dictionary is the *vector* dictionary (Table 7), which is organized like a detection dictionary, but failing vectors’ output responses are omitted. Another variation on the detection dictionary is the *list* dictionary, where only the outputs displaying errors are listed for every failing vector (rather than exhaustively enumerating all outputs).

Unlike the full response dictionary, detection, vector, and list dictionaries cannot be encoded in a simple bit-packed manner; the full response dictionary’s regular array of ones and zeros is now irregular and interspersed with integers that represent vector or output numbers. The size of a detection or vector dictionary is related to the number of fault detections stored in the dictionary. If the dictionary contains D detections, a simple vector dictionary encoding will consume at least $D \cdot \lceil \lg V \rceil$ bits, and the corresponding detection dictionary will require at least $D \cdot \lceil \lg V \rceil \cdot O$ bits. The size of the list dictionary will be given at the end of this section.

Boppana, Hartanto, and Fuchs have suggested a family of organizations based on *diagnostic trees* [1, 6]. A diagnostic tree gives the equivalence classes that are distinguished after each test is applied. The diagnostic tree for the example appears in Figure 2. When equivalence classes are large, this approach is concise because it allows for information about all of the faults in an equivalence class to be stored together in a single entry. Compaction can again be achieved by dropping faults when they are fully distinguished. For example, information about Fault 1 does not need to be recorded after the second vector.

Boppana and Fuchs have introduced two methods for encoding diagnostic tree organizations [5]. Encoding *DC1* stores the tree in two tables; the first stores an entry for the tree nodes. Each node contains the name of its

Fault	Detections
1	1:10 2:10 4:10;
2	1:00 4:00;
3	1:00 3:00 4:00;
4	1:01 3:00;
5	1:01 3:01;
6	1:01 3:01;
7	1:10 3:10 4:00;
8	2:11 4:11;

Table 6: The detection organization.

Fault	Detections
1	1 2 4;
2	1 4;
3	1 3 4;
4	1 3;
5	1 3;
6	1 3;
7	1 3 4;
8	2 4;

Table 7: The vector organization.

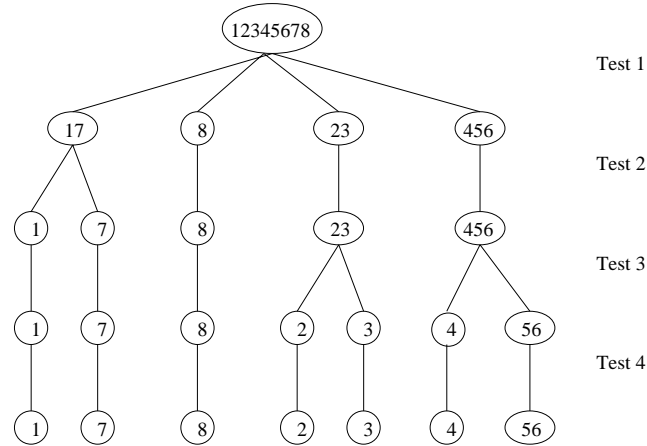


Figure 2: The diagnostic tree.

parent node and the output sequence leading from the parent node. The second table contains the name of the leaf node corresponding to each fault. Encoding *DC2* adds a third table—a fault-vector matrix that contains a 1 whenever a vector helps to further distinguish a fault. The addition of this table reduces the potential size of each node entry in the first table. The size of all of the diagnostic tree dictionaries is related to the rate at which faults are distinguished by the dictionary—a dictionary with poor diagnostic expectation will generally be smaller.

All of the organizations discussed thus far (with the exception of the diagnostic trees) have followed much the same high-level format: they all give faults as their primary axis of enumeration. For each fault, the organization provides a set of vectors, and for each vector a set of outputs. Vectors or outputs can be listed exhaustively (a fixed-length representation) or only called out explicitly when the fault is detected (an irregular list representation). Compaction may be achieved by omitting some part of the vector or output information.

Boppana, Hartanto, and Fuchs observed that this standard organization is not the only possibility [6]. Vectors or outputs could also be used to serve as the primary axis of enumeration. In fact, any combination of vectors, outputs, and faults could be enumerated exhaustively with the remaining axes given in list format. This observation leads to eight possible dictionary organizations, each of which is named by the axes that are listed exhaustively followed by the axes that are enumerated in list form. For example, the (vof)-() organization is equivalent to the full response dictionary: all axes are enumerated exhaustively. The the (f)-(vo) organization is equivalent to the list dictionary, where vectors and outputs are given in list format.

Of the eight organizations, the one that yields the smallest dictionary depends on the specific circuit and test set under consideration as well as the encoding used in conjunction with the organization. However, Boppana, Hartanto, and Fuchs showed that for the ISCAS benchmark circuits, exhaustively enumerating either vector-fault pairs (the (vf)-(o) organization) or output-fault pairs (the (of)-(v) organization) leads to the smallest dictionary

Fault	Entry
1	2 : 1 : : 1 2 :
2	1 2 : : : 2 :
3	1 2 : : 1 2 : 2 :
4	1 : : 1 2 : :
5	1 : : 1 : :
6	1 : : 1 : :
7	2 : : 2 : 2 :
8	: 1 2 : : 1

Table 8: The (vf)-(o) organization.

most of the time. Table 8 shows the (vf)-(o) organization for the example. Note that instead of explicitly mentioning any faults or vectors, the entry for each fault contains only a delimited list of faulty outputs for each vector.

The size of these dictionaries depends upon not only the number of faults, outputs and vectors, but also upon the number of *observed errors* (fault-vector-output triplets) the dictionaries store. Using the Bopanna, Hartanto, Fuchs encoding in which a single bit is used to represent each entry delimiter and E is the number of observed errors, the size of the (vf)-(o) dictionary is $E \cdot (\lg O + 1) + V \cdot F$ bits, the size of the (of)-(v) dictionary is $E \cdot (\lg V + 1) + O \cdot F$ bits, and the size of the list dictionary is $F \cdot (\log(V \cdot O)) + F$.

In Section 6 we show that these equations represent conservative estimates of the size of the dictionary under a Huffman encoding.

3 Diagnosing unmodeled faults

Aitken and Maxwell argue for diagnosing realistic faults directly by making a dictionary comprised of realistic fault signatures [4], but this approach has several weaknesses: The number of realistic faults in a circuit is significantly larger than the number of single stuck-at faults for a circuit, and the cost of simulating an individual realistic fault is usually much greater than the cost of simulating a stuck-at fault [9]. Furthermore, accurate modeling of realistic faults is critical to being able to diagnose them directly. A simulator that does a good job of fault grading may not have the accuracy required to produce a useful realistic fault dictionary. Finally, it is unlikely that a fault dictionary could be made to contain a behavior for every type of realistic fault under all conditions. Just as it is important to consider the coverage of unmodeled faults when examining fault coverage [19], it is important to address the performance of a fault dictionary with regard to faulty behaviors it does not contain.

It has been popular to investigate dictionary compaction methods that reduce the amount of information in the dictionary but do not affect the sizes of the equivalence classes contained in the dictionary. How much does eliminating this supposedly redundant information reduce the ability of the dictionary to diagnose unmodeled faults?

In order to answer this question, we performed the following experiment. First, we created four stuck-at fault dictionaries for each of the ISCAS-85 benchmark circuits:

1. A full dictionary.
2. A drop-when-distinguished dictionary.
3. A drop-on-10 dictionary.
4. A drop-on-5 dictionary.

Figure 3 shows the diagnostic expectation of these dictionaries. Note that the diagnostic expectation of the full dictionary and the drop-when-distinguished dictionary are identical, because the drop-when-distinguished dictionary does not increase the diagnostic expectation for the faults in the dictionary. As expected, the drop-on-10 dictionary has a larger diagnostic expectation than the full dictionary, and the drop-on-5 dictionary is worse than any of the other dictionaries. The drop-on-5 and drop-on-10 dictionaries roughly track the size of the drop-when-distinguished dictionary, though the drop-on-5 dictionary is the smallest for all circuits.

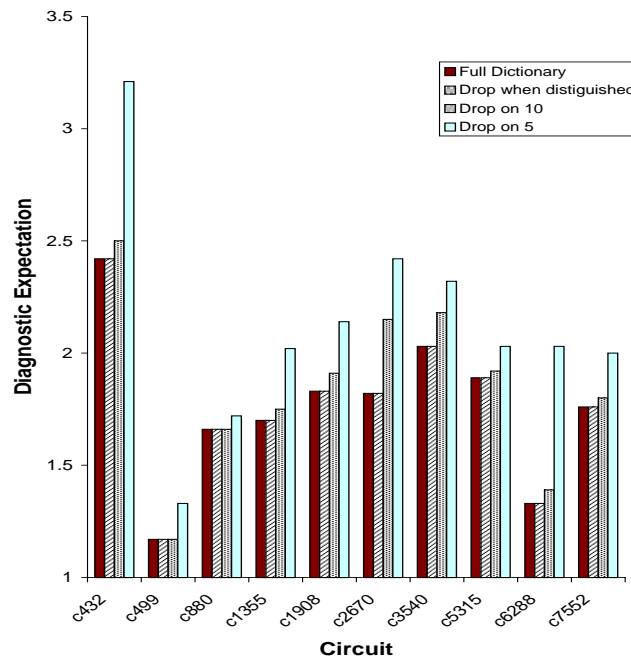


Figure 3: Diagnostic expectation for four dictionaries. The diagnostic expectation for the drop-when-distinguished dictionary is identical to the diagnostic expectation for the full dictionary.

Next, the Nemesis bridging fault simulator produced the observed faulty behaviors used to evaluate the dictionaries [8]. The behavioral model used by Nemesis for bridging faults is two-component simulation, in which the gates driving the bridged nodes are SPICE simulated to determine the bridge voltage, which is then compared against the SPICE-computed logic thresholds of downstream gates in order to model the effect of the Byzantine Generals Problem for bridging faults [2, 3]. The simulator also extensively models feedback bridging faults. If a feedback bridging fault evidences the potential to oscillate or hold state, the simulator forces the bridge voltage to be the fault free value on the bridged node closest to the primary inputs, which will disallow oscillation².

Because the faulty responses being diagnosed are not found in the dictionaries, the diagnosis procedure needs a method for determining which wires are involved in the targeted fault. That is, if wire A and wire B are bridged together, the procedure should identify that the fault involves wire A, wire B, or both. This experiment was modeled after the Teradyne fault diagnosis system [16], which penalizes candidate faults for each predicted

²Although a sensitized feedback loop created by the presence of a bridging fault can result in oscillation or state-holding behavior, most of the time the back node will overpower the front node. Rather than perform a SPICE simulation of each feedback-influenced region, we chose to create a simpler model.

failure that did not actually occur (*misprediction*) and for each failure that does occur without being predicted by the candidate (*nonprediction*). The procedure produces a ranked list of faults. Given this ranking, if any of the four possible stuck-at faults associated with the two bridged nodes appears among the ten highest ranked faults, the diagnosis is considered a success, otherwise it is considered a failure.

We did not use a matching technique that targets bridging faults explicitly [13] because we wanted to design an experiment using a very simple and commonly used algorithm that was blind to the type of defect being diagnosed. For instance, if the faults being diagnosed were breaks rather than bridges, the experimental design would not change.

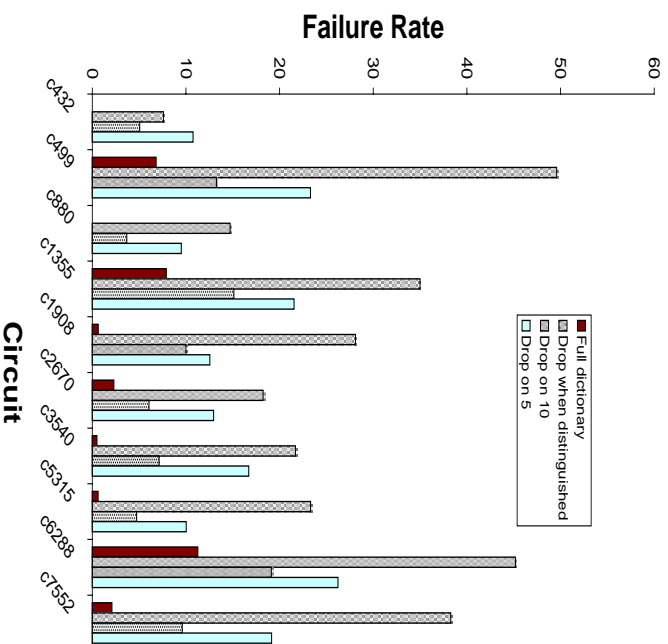


Figure 4: Diagnostic failure for the four dictionaries.

Finally, we attempted to diagnose signatures for the 10% of realistic bridging faults determined to be most likely to occur by the inductive fault analysis tool Carafe [11]. The results of the experiment are shown in Figure 4, which plots the percentage of diagnostic failures for each of the circuits using each of the dictionaries (note that the percentage of diagnostic failures for the full dictionary used on the c432 and the c880 is zero). As expected, the full dictionary always has the fewest diagnostic failures, followed by the drop-on-10 dictionary and the drop-on-5 dictionary. Surprisingly, the drop-when-distinguished dictionary has the most failures for all circuits except the C432. Note that the smallest dictionary, drop-on-5, did not have the most failures, so performance is not a function of dictionary size.

Figure 3 makes it appear that the full dictionary and the drop-when-distinguished dictionary have the same diagnostic ability—and that is true when the realm of diagnosis is restricted to single stuck-at faults—but the drop-when-distinguished dictionary is much less effective when applied to faults that it does not model. Because the standard metrics mask this shortcoming, we have no reason to believe that any other compaction scheme will do a better job. This argument forms the motivation for our exploration of lossless dictionary organizations in the following sections.

Fault	Entry
1	1 : 2 : 2 : 1 : 4 : 1 2;
2	1 : 1 2 : 4 : 2;
3	1 3 : 1 2: 4 : 1;
4	1 : 1: 3 : 1 2;
5	1 3 : 1;
6	1 3 : 1;
7	1 3 4 : 2;
8	2 : 1 2 : 4 : 1;

Table 9: Dictionary organization via vector set-error set pairs.

4 The Error Set Organization

If it is to retain all diagnostic information available, an organization must store the response of every fault to every test. The full dictionary achieves this goal in an inefficient manner; many entries are identical to the fault-free output. The detection organization overcomes this deficiency by storing only the output patterns for which the fault is detected, but this representation is also inefficient: Many of the bits in each output pattern do not carry discrepancies, and the same faulty output pattern may be repeated many times. The ideal organization will only store the difference between the fault-free circuit and each of the faulty circuits; additionally, when faulty circuits behave similarly, the ideal organization will either capitalize on this and store even less information (as done by the diagnostic tree organization), or allow the dictionary encoding to take advantage of the similarities between behaviors.

Our organization is based on *error sets*. An error set is a set of primary outputs that carry errors. As such, error sets are not specific to input patterns, output patterns, or faults. In order to create a specific faulty output pattern using an error set, the error set is simply associated with a fault-free output pattern. The faulty output pattern is then the fault-free output pattern with inverted values in the positions specified by the error set.

Our motivation for the new dictionary organization using error sets is that a fault often causes the same error set for many vectors. An organization could store the complete response of a fault to the test set by storing vector-error set pairs, which corresponds to the (f)-(vo) organization outlined by Bopanna, Hartanto, and Fuchs. The (f)-(vo) organization is a natural choice for adoption by a system that uses PPSFP fault simulation [18] because it follows the same flow as the fault simulator. But because error sets are reused so often, the basic entry for a fault in the error set dictionary is a set of vectors followed by an error set. A fault signature can be comprised of several such vector set-error set pairs. Note that a PPSFP fault simulator can still be used to produce the vector sets, but the information for a fault must not be recorded until the simulation of all vectors is completed. Table 9 shows the error set organization for the example data; it uses two delimiters: the first signals the switch between types of sets, and the second signals the beginning of a new fault signature. The entry for Fault 3 should be read as "Vectors 1 and 3 cause errors on Outputs 1 2, and Vector 4 causes an error on Output 1".

The error set dictionary format incorporates unknowns by extending the definition of an error set: a special symbol is used as a prefix to distinguish outputs that carry unknowns from outputs that carry errors. Thus, the error set dictionary has no problem recording information for circuits that lack reset states or with fault models that can potentially generate unknowns such as bridging or break faults. This mechanism will also work to allow the use of other alphabets for the purpose of producing dictionaries for delay faults or transition faults.

The error set organization allows for rearranging the information in a fault signature so that some vectors are associated with more than one error set. This means that the response for a specific vector is the union of each of the error sets specified for that vector. In some cases, rearrangement would allow for greater exploitation of the power of the error set format: the ability to associate multiple vectors with multiple failing outputs. However, the transformation of a fault signature into the smallest possible error set signature is a version of the

set-cover problem. In order to keep our implementation simple, we did not attempt to take advantage of this transformation.

One of the primary motivations for the dictionary format using error sets is the desire to include all diagnostic information. But, if the need for compaction is critical, error set dictionaries are amenable to compaction. Since the error set format is detection-based, it can easily make use of schemes that limit the number of detections recorded for each fault, such as drop-on-K or drop-when-distinguished.

Unlike many of the organizations reviewed in Section 2, the number of symbols in an error set dictionary cannot be derived solely from the circuit profile (number of outputs, vectors, etc.). Section 6 shows via empirical evidence that the number of symbols in an error set dictionary is related to the number of errors contained in the dictionary.

We recognize that the same error set is provoked by many different faults, and that there is frequently a great deal of similarity between error sets for the same fault. Instead of trying to capitalize on these redundancies in the error set organization, we leave them to be exploited by the dictionary encoding.

5 Encoding the error set organization

The simplest and most common approach to dictionary encoding is to directly translate the dictionary organization. When the organization requires that vector or fault numbers be recorded, a simple encoder will devote a fixed number of bits to each entry. Delimiters like end-of-fault or end-of-vector markers can also be assigned a fixed number of bits. If the organization calls for the output response of the circuit, the encoding may store output strings in a bit-packed manner.

Rather than propose an encoding for the error set organization, we chose to explore the use of standard data compression techniques. For our purposes, useful encoding algorithms must be usable for both creating and using dictionaries. When creating dictionaries, encoding algorithms must be fast with respect to the fault simulator producing the data, they cannot require multiple passes over the data (because this would require diagnostic fault simulation to be performed more than once), and they must not require all the data to be present before compression begins—it may not be possible to store the dictionary in its uncompressed form. When using the dictionary for diagnosis, the encoding algorithms must allow the dictionary to be decoded in sections—it should not be necessary to decode the end of the dictionary in order to access the beginning of the dictionary. After a decoded section is used by the diagnosis algorithm, it can be overwritten by the next section. Additionally, while not a requirement, it would be convenient if a section in the middle of the compressed dictionary could be decoded without decoding preceding sections. Being able to decode a middle section of the dictionary is not important for the standard scheme of comparing an observed response to every signature in the dictionary, but it is useful for looking up the behavior of a particular fault or set of faults.

The most widely used data compression algorithms that meet the requirements are the Lempel-Ziv family of sequential data compressors [21, 22] (here sequential does not refer to a type of circuit, but to a compression algorithm that identifies repeated sequences). It might also be feasible to use an entropy encoder such as arithmetic coding or Huffman coding, but either an adaptive form of the encoder must be used, or the data must be processed in sections in order to meet the one pass requirement detailed above. For this research, we chose to investigate the use of Lempel-Ziv-Welch (LZW) compression [20] and Huffman coding.

In addition to compression algorithms, there exist transformations that make the data more amenable to compression. The first of two such transformations that we found useful is one created by Burrows and Wheeler [7], who suggest a clever reversible sort followed by a move-to-front transformation (Burrows and Wheeler recommend the transformation should be followed by an entropy encoder). Burrows-Wheeler works well with the error set organization because error sets and portions of error sets are shared between many dictionary entries.

The second, differential encoding, follows from the observation that error sets can be written as an ascending sequence of integers. After the transformation, instead of listing outputs or vectors, each set consists of an initial value followed by a sequence of differences from the previous value. In this way “1 2 3 4” and “4 5 6 9” transform

to “1 1 1 1” and “4 1 1 3”. The increased repetition will improve the performance of most data compression algorithms.

For any data compression algorithm, several decisions must be made in order to complete the implementation. When we report the results for LZW compression or Huffman coding, we first use differential encoding (except where noted). For any LZW implementation, the size of the table that records the symbol sequences is affected by the number of bits used to represent a codeword; we used sixteen bit codewords. When all of the codewords are used up (after using 2^{16} codewords), we empty the symbol sequence table and begin compression again. This is an inefficient choice (because it requires the algorithm to reidentify commonly used sequences), and it results in dictionaries as much as ten percent larger than they might have been otherwise, but it is simple to implement and has the advantage of allowing the compressed dictionary to be decoded in sections.

When we report the results for the Burrows-Wheeler technique, we process the data in 900 kilobyte blocks—following Burrows-Wheeler with Huffman coding. When we apply Burrows-Wheeler to the error set organization, we do not use differential encoding. We found that using differential encoding before using the Burrows-Wheeler operation actually increases the size of the resulting dictionary. Differential encoding degrades the effectiveness of the Burrows-Wheeler sort.

6 Results

This section reports results for 27 circuits and associated input sets. The first columns of Table 10 give statistics for the ten combinational ISCAS circuits, the larger sequential ISCAS circuits treated as fully scanned circuits, and some sequential circuits (treated as non-scan, non-resettable circuits). The test sets for the combinational and scan benchmarks were generated by the Nemesis ATPG system [12]. The test sets for the sequential benchmarks were generated by the HITEC test pattern generation system [14]. Because the sequential circuits were treated as non-resettable, there were many unknown output values in both the faulty and fault-free circuit responses. We present results only for sequential circuits with test sets that provide reasonable diagnostic capability (if very few of the faults in a circuit are detected, the diagnostic capability of the test set will be poor). For all of the results presented here, we assume that the dictionary must have the capability to store potential detects (because it is hard to be certain ahead of time that potential detects will not occur). The error set dictionaries use a special symbol prefix to record potential detects, as discussed in Section 5, and the (vf)-(o) and (f)-(vo) dictionaries have a second entry for each fault to record the potential detects, as suggested by Boppana, Hartanto, and Fuchs [6].

Tables 10 and 11 report exhaustive results for all circuits so that readers may verify our results and investigate additional hypotheses of their own, but for purposes of elucidation, Figures 5 through 10 illustrate important points of discussion by showing data (and least-squares-fit linear approximations for the data) for the six dictionaries recording the most observable errors (five of which are among the largest ISCAS-89 circuits considered as scan circuits, and the largest of which is the largest sequential circuit).

Of the lossless dictionary organizations discussed in Section 2, the (vf)-(o) organization produced the smallest dictionaries using the encoding originally suggested by Boppana, Hartanto, and Fuchs [6]. We will report compression results for the (vf)-(o) organization, for the error set organization, and for the (f)-(vo) organization (because it is an organization that naturally fits well into a system using a PPSFP simulator).

The last three columns of Table 10 give the number of symbols for each of these three organizations before encoding. Note that these are not dictionary sizes: until the dictionary has been encoded, its size cannot be determined. Table 11 reports the size in bytes of the three organizations under the encodings discussed in Section 5. For each organization, we encode using the suggestions of Burrows and Wheeler (labeled BWT), with LZW encoding (labeled LZW), and with Huffman coding (labeled Huffman). For the error set organization, we use Huffman coding with differential encoding (labeled differential) and without differential encoding (labeled natural).

Figure 5 plots the number of symbols for the largest six circuits shown as a function of the number of observed errors recorded in the dictionaries. The data points for the error set organization deviate little from their linear

circuit	Circuit characterization					number of symbols pre-encoding		
	errors	detections	POs	faults	vectors	Eset	(vf)-(o)	(f)-(vo)
c432	12 283	4 842	7	524	71	19 398	86 691	22 491
c499	16 723	14 803	32	758	56	38 575	101 619	47 087
c880	16 837	12 881	26	942	71	24 122	150 601	43 541
c1355	40 837	36 840	32	1 574	88	90 102	317 861	116 091
c1908	127 606	66 692	25	1 879	146	135 609	676 274	262 869
c2670	143 011	89 480	140	2 747	175	125 053	1 104 461	324 718
c3540	216 058	91 934	22	3 428	205	203 825	1 621 538	403 354
c5315	271 848	146 940	123	5 350	172	331 006	2 112 248	571 078
c6288	336 566	156 097	32	7 744	57	372 904	1 219 382	656 504
c7552	826 377	399 257	108	7 550	304	583 308	5 416 777	1 632 441
s5378scan	724 902	392 601	228	4 603	393	557 025	4 342 860	1 514 707
s9234scan	1 118 276	4 597 251	250	6 929	684	1 032 282	10 597 148	10 319 707
s13207scan	2 566 939	1 632 044	790	9 815	763	2 104 097	17 544 629	5 840 842
s15850scan	2 983 490	1 503 857	684	11 721	649	2 892 886	18 197 348	6 002 925
s35932scan	1 044 401	766 866	2 048	39 094	88	1 089 468	7 924 945	2 617 227
s38417scan	16 380 276	9 715 103	1 742	31 180	1 459	14 547 485	107 363 516	35 841 662
s38584scan	14 392 020	9 369 901	1 730	36 304	1 174	12 954 317	99 633 812	33 168 126
s298	52 342	21 599	6	308	259	26 860	211 886	95 848
s344	34 009	13 209	11	327	108	38 007	104 641	60 754
s641	26 908	20 459	24	465	211	35 197	223 138	68 291
s713	25 426	19 890	23	581	175	34 467	228 776	65 787
s820	174 493	95 773	19	861	968	138 818	1 841 389	366 900
s832	175 530	96 179	19	861	967	139 286	1 840 704	368 749
s1238	55 970	37 816	14	1 372	478	74 287	1 367 602	132 974
s1423	6 080	5 272	5	1 516	88	8 561	272 896	18 140
s5378	1 048 663	699 943	49	4 603	900	833 296	9 334 063	2 453 152
s35932	18 717 819	44 790 72	320	39 094	381	19 535 331	48 507 447	27 715 057

Table 10: Circuit characterization for each of 27 combinational and sequential circuits.

approximation, which lends credence to our assertion that the number of symbols in an error set organization grows linearly with the number of errors that it contains. Figure 5 might seem to imply that the (f)-(vo) organization is a good one because it produces fewer symbols than the (vf)-(o) organization, but Figures 7 and 8 show that the (vf)-(o) organization compresses better and so produces smaller dictionaries. This is because a large percentage of the symbols in the (vf)-(o) dictionary are separators rather than integers. The error set organization has a small number of symbols and, as Figure 9 shows, it is also amenable to compression.

Figure 6 shows the effectiveness of differential encoding used in conjunction with Huffman coding. Differential encoding also helped LZW compression, but there it only decreased the size of the resulting dictionaries by about ten percent (instead of the factor of two that achieved when used with Huffman coding).

As illustrated by Figure 7, the size of the (vf)-(o) dictionaries varies by more than 500%—depending on the encoding. Huffman coding always produces smaller dictionaries than the original encoding suggested by Boppana, Hartanto, and Fuchs. The Burrows-Wheeler encoding consistently produces the smallest dictionaries.

As illustrated by Figure 8, the size of the (f)-(vo) dictionaries only varies by about 100% under different encodings, but unfortunately, the size of the smallest dictionaries (Burrows-Wheeler encoded) is only a little bit smaller than the Huffman encoding of the (vf)-(o) organization: The Burrows-Wheeler encoding of the (vf)-(o) organization are about half of the size of the Burrows-Wheeler encoding of the (f)-(vo) organization.

Figure 9 shows that the size of the error set dictionaries vary over a small range, and as shown by Figure 10, the growth rate of the Burrows-Wheeler encoding of the error set organization is 15% smaller than the growth rate of the next best dictionary (the (vf)-(o) organization under Burrows-Wheeler encoding). In fact, the growth rates of both Burrows-Wheeler encodings are smaller than the growth rate of the bit-packed encoding of the pass-fail organization. Furthermore, of the four best organization-encoding pairs, three of them use the error set

circuit	Error set organization				(vf)-(o) organization			(f)-(vo) organization		
	BWT	LZW	natural	Huffman differential	BWT	LZW	Huffman	BWT	LZW	Huffman
c432	5417	11068	10779	9673	5255	8488	8115	6522	13822	11718
c499	11102	20052	24080	20482	10940	19792	18310	14334	33308	29318
c880	10658	17442	16950	13481	11701	18040	17919	14886	30984	26588
c1355	20821	42208	59826	53375	25978	45182	50035	28260	72616	75769
c1908	32213	64726	104236	77351	44863	77300	109245	56406	152568	186270
c2670	40628	66812	111716	68091	46817	83602	172078	81957	235062	243689
c3540	69468	113622	164123	117540	82597	114476	241800	95854	230910	275683
c5315	135964	214630	272417	218187	141112	215480	279751	212878	452456	451454
c6288	94385	168660	265838	180725	75587	123152	160784	105001	302774	364770
c7552	167429	302316	576240	313339	256168	445458	989291	452945	1048638	1221151
s5378scan	210550	318456	577367	327607	292891	515684	1098965	612646	1329472	1288449
s9234scan	380492	582378	1146028	607359	517635	905476	2098694	1267664	2485150	2279119
s13207scan	758979	1068608	2525374	1048025	1014488	2013874	3922970	3458805	6264728	5734855
s15850scan	1119395	1617712	3244537	1676394	1326722	2455698	4677663	3295643	6046162	5955705
s35932scan	404080	592800	1045157	630743	807666	1447430	2104183	1375767	2692122	2619029
s38417scan	7706853	9387090	18669436	8680639	8263086	14451918	22551843	24981088	40387650	38791294
s38584scan	6015074	7192846	17091725	6734761	7038252	12401500	27523622	22459840	37949380	36353904
s298	5177	12130	25665	12072	6629	16168	33559	12825	49108	55774
s344	10519	22418	25543	20509	8588	20442	23190	14062	35660	36211
s641	14145	24572	29986	22180	16059	27986	50057	23621	56440	49603
s713	12181	23204	29169	21485	14808	27120	48087	19830	51058	45983
s820	41151	74996	152563	91687	53990	93254	215568	102019	262226	287881
s832	41422	75808	153167	93347	55087	94660	214607	104146	265378	279215
s1238	28538	48056	66147	46874	39988	51664	111501	45464	96562	97036
s1423	2439	5702	6159	4510	3650	7266	9457	3646	10868	10203
s5378	189840	322012	1003961	455829	299357	453152	1184289	604123	1691776	2053547
s35932	6153999	9719068	18947852	12869015	6353708	10256472	16364092	10834793	18152996	20293865

Table 11: Compression results, shown in number of bytes, for 27 circuit-input set combinations using the error set, (vf)-(o), and (f)-(vo) organizations and Burrows-Wheeler followed by Huffman coding, LZW encoding, and Huffman encoding.

organization.

The run times for the compression methods that we tried were all small compared to the time for diagnostic simulation. Encoding the error set organization using the Burrows-Wheeler transformation took four minutes for the non-scan s35932 using a 300 MHz PentiumII. Encoding using the Burrows-Wheeler transformation took much longer on the (vf)-(o) organization (more than an hour for the s35932); this is because the long runs of repeated symbols play havoc with the Burrows-Wheeler sort. It may be fixable by using a run-length encoder before the Burrows-Wheeler transformation.

7 Conclusions and future work

Regardless of the technique, removing information from a dictionary will cause it to do a poorer job of diagnosing faults that were not used to construct the dictionary, as shown by the experiment presented in Section 3. This observation is particularly important when considering compaction techniques that do not change the diagnostic expectation of a dictionary: the implication that the usefulness of the dictionary has not been reduced by compaction is misleading. Current dictionary quality metrics are inadequate. Future research must supply quality metrics that are useful when considering the diagnosability of unmodeled faults.

Error set compression yields dictionaries that are often smaller and not computationally more expensive to produce than one of the smallest of the compacted dictionaries: the pass-fail dictionary. Pass-fail dictionaries are not adequate to many diagnosis tasks, and when more information is required, the error-set dictionaries are

consistently several times smaller than any previous full-information method.

Each of the error set dictionaries for the ISCAS-85 circuits is significantly smaller than the MCNC physical design information for the corresponding circuit. Using a full-information dictionary for diagnosis is practical: error set dictionaries are not prohibitively large or computationally expensive.

Using standard data compression techniques on fault dictionaries raises several questions to be answered by future research. First, we have only considered the use of lossless encoding techniques. Most previous research has focused on lossy organizations; could lossy encoding techniques, rather than lossy organizations, be applied fruitfully to fault dictionaries? How would lossy encoding affect the diagnosis of unmodeled faults?

Dictionaries are not just stored—they are used to diagnose faulty circuits. We are interested in tailoring dictionary storage to facilitate the diagnosis procedure. To that end, it would be useful to make approximate comparisons between fault signatures in their compressed form, and it would be useful to be able to take the union of compressed fault signatures in order to perform bridging fault diagnosis.

References

- [1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.
- [2] J. M. Acken and S. D. Millman. Accurate modeling and simulation of bridging faults. *Proceedings of the Custom Integrated Circuits Conference*, pages 17.4.1–17.4.4, 1991.
- [3] J. M. Acken and S. D. Millman. Fault model evolution for diagnosis: Accuracy vs precision. *Proceedings of the Custom Integrated Circuits Conference*, 1992.
- [4] R. C. Aitken and P. C. Maxwell. Better models or better algorithms? on techniques to improve fault diagnosis. *Hewlett-Packard Journal*, February 1995.
- [5] V. Boppana and W. K. Fuchs. Fault dictionary compression by output sequence removal. In *Proceedings of International Conference on Computer-Aided Design*, pages 576–579. IEEE, 1994.
- [6] V. Boppana, I. Hartanto, and W. K. Fuchs. Full fault dictionary storage based on labeled tree encoding. In *Proceedings of the VLSI Test Symposium*, pages 174–179. IEEE, 1996.
- [7] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Systems Research Center, 1994.
- [8] B. Chess and T. Larrabee. Logic testing of bridging faults in CMOS integrated circuits. *IEEE Transactions on Computers*, pages 338–345, March 1998.
- [9] B. Chess, T. Larrabee, and C. Roth. On evaluating competing bridge fault models for CMOS ICs. In *Proceedings of the 1994 VLSI Test Symposium*, pages 446–451. IEEE, 1994.
- [10] D. Feltham and W. Maly. Physically realistic fault models for analog CMOS neural networks. *IEEE Journal of Solid-State Circuits*, 26(9):1223–1229, September 1991.
- [11] A. Jee and F. J. Ferguson. Carafe: An inductive fault analysis tool for CMOS VLSI circuits. In *Proceedings of the IEEE VLSI Test Symposium*, pages 92–98, 1993.
- [12] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, pages 4–15, January 1992.
- [13] D. Lavo, B. Chess, T. Larrabee, and F. J. Ferguson. Diagnosing realistic bridging faults with single stuck-at information. *IEEE Transactions on Computer-Aided Design*, pages 255–268, March 1998.

- [14] T. Niermann and J. Patel. HITEC: A test generation package for sequential circuits. In *Proceedings of European Design Automation Conference*, pages 214–218, 1991.
- [15] I. Pomeranz and S. M. Reddy. On the generation of small dictionaries for fault location. In *Proceedings of International Conference on Computer-Aided Design*, pages 272–279. IEEE, 1992.
- [16] J. Richman and K. R. Bowden. The modern fault dictionary. In *Proceedings of International Test Conference*, pages 696–702. IEEE, 1985.
- [17] P. G. Ryan, W. K. Fuchs, and I. Pomeranz. Fault dictionary compression and equivalence class computation for sequential circuits. In *Proceedings of International Conference on Computer-Aided Design*, pages 508–511, 1993.
- [18] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy. Fault simulation for structured VLSI. *VLSI Design*, VI:20–32, 1985.
- [19] L. W. Wang, M. R. Mercer, T. W. Williams, and S. W. Kao. On the decline of testing efficiency as fault coverage approaches 100%. In *Proceedings of the 1995 VLSI Test Symposium*, pages 74–83. IEEE, 1995.
- [20] T. A. Welch. A technique for high-performance data compression. *IEEE Computer*, pages 8–19, June 1984.
- [21] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, pages 337–343, May 1977.
- [22] J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, pages 530–535, September 1978.

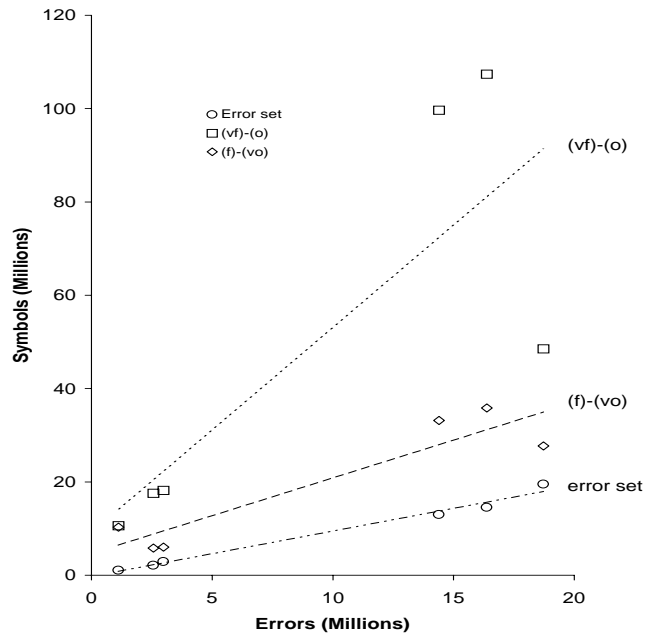


Figure 5: Number of Symbols for each organization

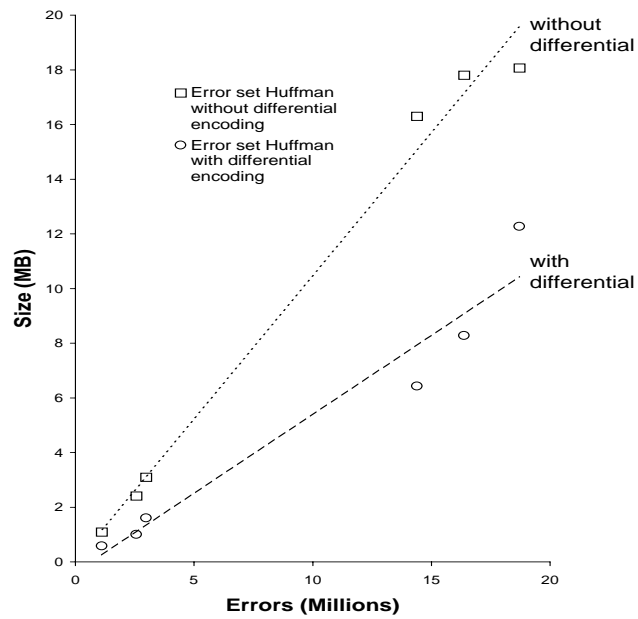


Figure 6: The effect of differential encoding on Huffman coding under the error set organization

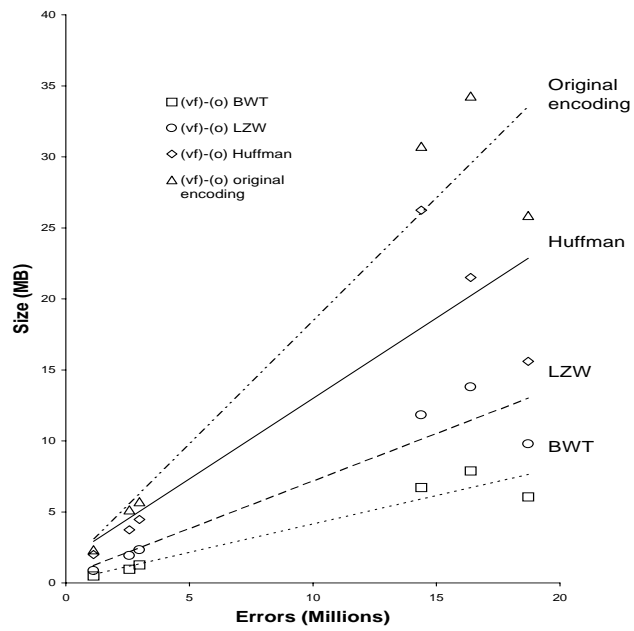


Figure 7: Different encodings under the (vf)-(o) organization

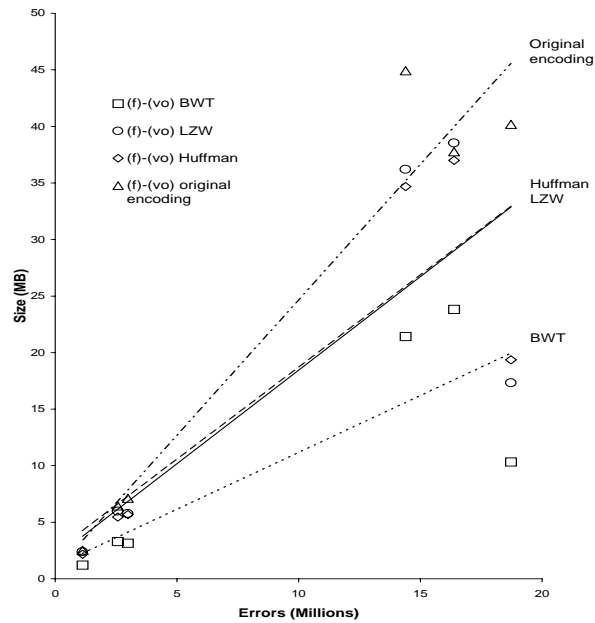


Figure 8: Different encodings under the (f)-(vo) organization

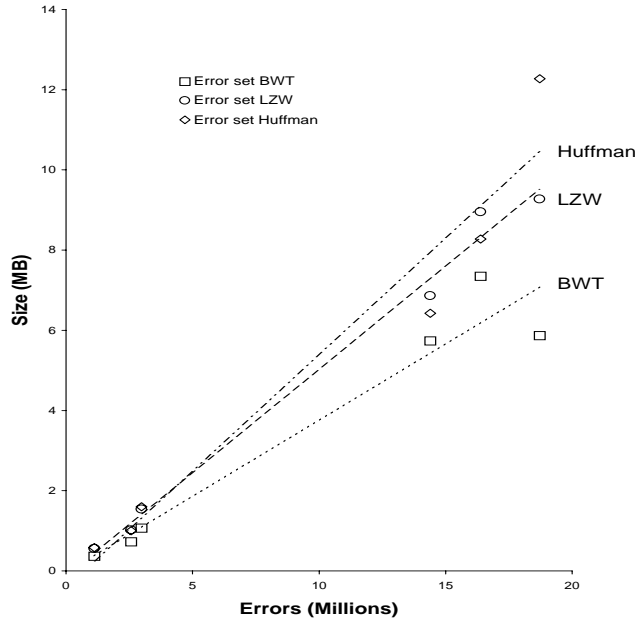


Figure 9: Different encodings under the error set organization

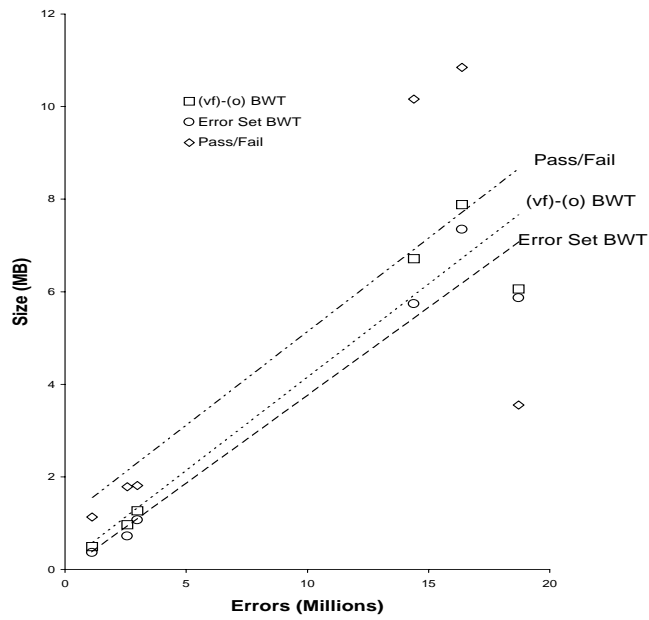


Figure 10: The smallest three dictionary formats