Fault Dictionary Size Reduction for Million-Gate Large Circuits

Yu-Ru Hong and Juinn-Dar Huang

Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan
yrhong.ee94g@nctu.edu.tw, jdhuang@mail.nctu.edu.tw

Abstract - In general, fault dictionary is prevented from practical applications for its extremely large size. Several previous works are proposed for the fault dictionary size reduction. However, they might not be able to handle today's million-gate circuits due to the high time and space complexity. In this paper, we propose an algorithm to significantly reduce the size of fault dictionary while still preserving high diagnostic resolution. The proposed algorithm possesses extremely low time and space complexity by avoiding constructing the huge distinguishability table, which inevitably boosts up the required computation complexity. Experimental results demonstrate that the proposed algorithm is fully capable of handling industrial million-gate large circuits in a reasonable amount of runtime and memory.

I. INTRODUCTION

Semiconductor technology progresses constantly so that electronic products with higher performance and various novel functions can be provided to end users. However, denser and finer fabrication processes make chips vulnerable to defects, and hence lower the manufacturing yield and reliability. Fault diagnosis techniques are then proposed to help locate faults and determine the causes of failures in the manufacturing process. Based on the diagnosis results, manufacturers can refine their processes while designers can modify their designs to improve the yield of chips.

Conventionally, diagnosis techniques are classified into two major categories, effect-cause and cause-effect analysis [1]. Dynamic diagnosis is an effect-cause analysis which observes the faulty response of the circuit under test (CUT) and deduces the cause of error based on the fault-free response. There are many research works devoted to dynamic diagnosis [2-4]. For the cause-effect analysis, diagnosis uses a pre-computed fault dictionary generated through fault simulation. The dictionary stores the faulty output response of the CUT in the presence of every modeled fault. By comparing the response of the CUT to the response in the dictionary, faults can be recognized and hence located. If, however, in the presence of different faults, the CUT responses the same to any test vector in a given test set, these faults form an equivalence class which limits the maximum diagnostic resolution of the test set. Here the diagnostic resolution (DR) is defined as the fraction of distinguishable faults of all modeled faults.

Nevertheless, two obstacles generally prevent fault dictionary to practical applications: (1) Creating a dictionary with high diagnostic resolution requires a long computation time. (2) The size of dictionary is extremely large and thus impractical to use. The first obstacle has been evaluated by the authors in [5]. They show that only a small number of diagnostic runs are sufficient to compensate for the nonrecurring effort of creating a fault dictionary. Therefore, fault dictionary is still attractive as long as its size can be kept small. Unfortunately, as the circuit size increases, the memory requirement of fault dictionary grows so rapidly and thus becomes unacceptable.

Though many research works [6-8] are contributed to overcome this obstacle, some of them are still unable to bring down the dictionary size to an acceptable level; others consume too much time and space to be applied on real industrial circuits. In [6], the proposed method requires to build the *distinguishability table* whose space complexity is up to $O(|T|*|F|^2)$, where $|T|$ is the number of test vectors and $|F|$ is the number of faults. The complexity is simply too high to be acceptable for million-gate circuits. In this paper, an improved algorithm is proposed in which there is no need to construct the distinguishability table. As a result, our algorithm possesses extremely low time and space complexity. The experimental results exhibit the excellent runtime efficiency, low memory space requirement and great dictionary size reduction capability of our method.

The rest of this paper is organized as follows. Section II briefly describes several related works. Section III introduces some background knowledge of our work. Section IV presents the proposed method. Experimental results and conclusions are given in Section V and VI, respectively.

II. RELATED WORKS

A full fault dictionary stores the full response for each test vector applying to the CUT. It possesses the largest size of all kinds of fault dictionaries and its size is $O(R*|T|*|F|)$ for a dictionary of R primary outputs, $|T|$ test vectors, and $|F|$ faults. Other forms of dictionaries such as the pass-fail dictionary and the vector dictionary are all compacted or compressed by certain approaches. These approaches can be further partitioned into two groups — using lossless or lossy techniques.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

Lossless methods preserve the same diagnostic resolution as the full dictionary. One popular lossless compaction technique is to store merely the responses of the failed test vectors for each modeled fault. Another research work develops alternative storage structures by encoding [9]. On the other hand, lossy approaches may sacrifice some diagnostic resolution to achieve an even smaller dictionary size. A *pass-fail dictionary* does not store the faulty output responses of the CUT at all. Instead, it contains an entry for each fault-test pair to indicate whether the test vector can detect the fault or not. Usually, '0' stands for an undetected fault; '1' stands for a detected fault. The size of pass-fail dictionary is O(|T|*|F|) only. Unlike the pass-fail dictionary, a *vector dictionary* alternatively stores the indices of the failed test vectors. Among various exploited compaction techniques, pass-fail and vector dictionaries are comparatively simple and small. However, the dictionary size is still too big to be applied to today's multi-million-gate circuits even after the compaction. Hence, it is an appealing tradeoff to tolerate slight degradation of diagnostic resolution for drastically smaller memory storage requirement.

In [6], on the basis of pass-fail dictionary, authors propose a size reduction method which partitions the test set and stores a combined signature for each partition. With a minor loss in diagnostic resolution, the dictionary can be compacted to the ideal $\lceil \log_2|F| \rceil$ partitions. Thus, the size of the fault dictionary is reduced to only $\log_2|F|/|T|$ times large. The compaction process, however, needs to construct a distinguishability table with the space complexity O(|T|*|F|^2), which makes its time complexity at least the same level high. Therefore, it is highly doubtful whether the method of such high time and space complexity can be applied to today's SoC chips with millions of possible faults.

III. PRELIMINARIES

Given a fault set F and a test vector set T, a pass-fail dictionary D is defined as a |T|x|F| matrix where

$$\begin{cases} D_{ij} = 1, \text{if } t_i \text{ can detect } f_j & 1 \le i \le |T|, 1 \le j \le |F| \\ D_{ij} = 0, \text{otherwise} & 1 \le i \le |T|, 1 \le j \le |F| \end{cases}$$

Fig. 1 shows an example of a pass-fail dictionary D1. For example, the test vector t_1 can detect two faults f_4 and f_5, while the test vector t_2 can detect f_2, f_3, f_4 and f_5. Based on the pass-fail dictionary, we construct a subset of T, the selected test set T_S, in which are the vectors to be stored in the resultant reduced fault dictionary. That is, our proposed method aims at efficiently constructing a small but effective T_S. In this paper, T_S is considered as an ordered sequence for convenience in the vector selection phase, though the order does not actually affect the diagnostic resolution of the selected test set.

Once the T_S is created, the corresponding fault equivalence sets are formed. A fault equivalence set F_E is a set of faults in which all fault pairs are indistinguishable with respect to T_S. Given D1 in Fig. 1, assume $T_S = \{t_1\}$, then there are two fault equivalence sets, $F_{E1} = \{f_1, f_2, f_3\}$ and $F_{E2} = \{f_4, f_5\}$. The distinguishability information can be presented using a table by first defining a fault pair set P as

$$P = \{ fp_k = (f_i, f_j) \mid f_i, f_j \in F, i < j \}.$$

Then, the distinguishability table A is a |T|x|P| matrix, where

$$\begin{cases} A_{ij} = 1, \text{if } t_i \text{ can distinguish } fp_j \\ A_{ij} = 0, \text{otherwise} \end{cases}$$

A1 in Fig. 2 is the distinguishability table obtained from D1 in Fig. 1. The row of t_1 in A1 with 4 0's which indicate fault pairs (f_1, f_2), (f_1, f_3), (f_2, f_3), (f_4, f_5) are indistinguishable for t_1. Now, the dictionary size reduction problem can be translated to finding a minimum number of rows to cover all fault pairs. This set-cover problem is a well-known NP-complete problem [10]. Even worse, due to the extremely high space complexity O(|T|*|F|^2) of distinguishability table, a circuit with one million faults leads to a table whose size is of 10^{15} order even if only 1000 test vectors are present.

A sub-optimal but simple heuristic algorithm is therefore necessary to deal with this problem. As in [6], a greedy algorithm is adopted. This algorithm simply selects the best test vector which distinguishes most new fault pairs in every iteration. Even though the greedy algorithm is so simple, the problem remains intractable for an extremely large distinguishability table. Take A1 as an example to show how this algorithm works. In the 1st iteration, t_1, t_3 and t_4 can distinguish 6 fault pairs, while t_2 distinguishes only 2 fault pairs. So the algorithm arbitrarily selects t_1 into T_S. In the 2nd iteration, it skips the 6 recognized fault pairs by t_1 and evaluates the remaining test vectors. t_3 outperforms others by distinguishing 3 new fault pairs and is added to T_S. Then, all fault pairs are covered after t_2 is selected in the last iteration. The algorithm ends with $T_S = \{t_1, t_3, t_2\}$. This greedy set-cover algorithm is shown in Fig. 3. The time complexity of Greedy-Set-Cover algorithm is O(N*|T|*|F|^2), where N is a given upper bound of the size of T_S. Ideally, when test vectors evenly partition the faults into detected and undetected ones, the minimal possible test size is $\log_2|F|$, which is generally hard to achieve since a typical test vector usually detects only a small portion of faults and leaves most faults undetected. Though the greedy algorithm is simple, the O(|T|*|F|^2) introduced by the distinguishability table still keeps the algorithm far from practical for large circuits.

	f_1	f_2	f_3	f_4	f_5
t_1	0	0	0	1	1
t_2	0	1	1	1	1
t_3	1	1	0	0	1
t_4	0	1	1	0	0

Fig. 1. Pass-fail dictionary D1.

	(f_1,f_2)	(f_1,f_3)	(f_1,f_4)	(f_1,f_5)	(f_2,f_3)	(f_2,f_4)	(f_2,f_5)	(f_3,f_4)	(f_3,f_5)	(f_4,f_5)
t_1	0	0	1	1	0	1	1	1	1	0
t_2	1	1	1	1	0	0	0	0	0	0
t_3	0	1	1	0	1	1	0	0	1	1
t_4	1	1	0	0	0	1	1	1	1	0

Fig. 2. Distinguishability table A1.

```
Greedy-Set-Cover(F, T, N) {
  Ts ← ∅
  do
    for i ← 1 to |T|
      if ti ∉ Ts
        Yi ← Ts ∪ {ti}
        calculate CFPi
    identify tj with the largest CFP
    Ts ← Ts ∪ {tj}
  while (uncovered fault pairs exist
         and |Ts| < N)
  return Ts
}
```

N is the given upper bound of the number of selected vectors
T_S is the set of selected test vectors
Y_i is a temporary test set
CFP_i is the number of covered fault pairs

Fig. 3. The pseudo code of Greedy-Set-Cover algorithm.

IV. Proposed Method

In this section, we start by translating the distinguishability table into its graph equivalent and identify several important properties which serve as the keystones in the proposed algorithm.

First, construct a complete graph G with |F| vertices. Each vertex v_i represents for the fault f_i. Then, for a given selected test set T_S and its corresponding distinguishability table, remove the edge (v_i, v_j) in G if (f_i, f_j) is covered by a test vector in the distinguishability table, That is, the remaining edges in G represent those indistinguishable fault pairs under T_S. The resultant graph has the following properties:

(1) Graph G is a complete graph if $T_S = \varnothing$.

(2) There is an edge between (v_i, v_j) if and only if the fault pair (f_i, f_j) is indistinguishable under the given T_S.

(3) T_S partitions G into a set of disjoint connected components. Faults within the same connected component form a fault equivalence set.

(4) Each connected component, representing a fault equivalence set, is a complete graph.

Property (1) is trivial since an empty test set is incapable of removing any edge from G. Property (2) directly comes from the operations of edge removal; it implies that the edges represent those uncovered 0's in the distinguishability table. Property (3) says that after removing specific edges according to the distinguishability table, G is broken into a set of disjoint connected components. Because a test vector can only distinguish the detected faults from the undetected ones, it actually applies a cut on the graph G. Edges across the cut are then removed. The remaining edges represent the indistinguishabiliy among the faults within a connected component, forming a fault equivalence set. Property (4) is based on the definition of fault equivalence set, in which every two faults are indistinguishable. In other words, there should be edges between all vertex pairs within a connected component. Hence every connected component in G must be a complete graph. The above properties can be examined using the example shown in Fig. 4, which is derived from the same example given in Section III. The initial graph is complete, as property (1) states. Then the growing test set continually partitions the graph into more disjoint complete connected components until all fault pairs are distinguishable. Property (2), (3) and (4) can be clearly observed from the graphs shown in Fig. 4.

Next, we reexamine the greedy algorithm introduced in the previous section in a perspective of graph. The greedy algorithm always selects the best test vector to cover most fault pairs in an iteration. In graph, the best vector is actually the one which removes the most edges if it is added into the current T_S. Fig. 5 gives an example to demonstrate the effectiveness of two different vectors. There are two candidates, t_1 and t_2, to be added into the selected test set. However, t_1 can remove 6 edges while t_2 can remove only 4 edges from the initial complete graph. Thus t_1 is a better choice than t_2 in this case. There is also an example in Fig. 4 to illustrate the overall selection process of the previous greedy algorithm. In fact, removing most edges is equivalent to minimizing remaining edges. Therefore, the optimization target can be transformed from finding a minimum set-covering of the distinguishability table into minimizing the remaining edges in the corresponding graph.

Based on the above properties, the number of remaining edges in the graph for a given selected test set T_S can be calculated as follows. Assume T_S partitions all faults into n fault equivalence sets, F_{E1}, F_{E2}, F_{E3}, …, F_{En}. From Property (4), each fault equivalence set is represented as a complete

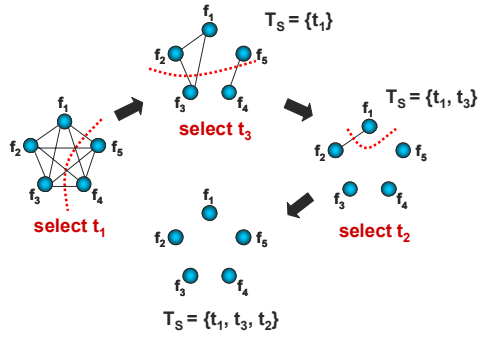

Fig. 4. Selection process resulting in a set of complete connected components.

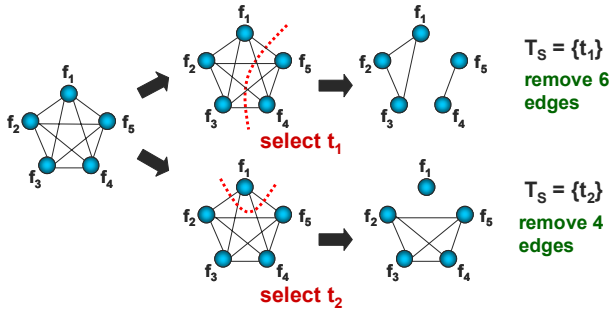

Fig. 5. Illustration of effectiveness of different test vectors.

connected component. Hence, the number of remaining edges in the graph is

$$\binom{|F_{E1}|}{2}+\binom{|F_{E2}|}{2}+\cdots+\binom{|F_{En}|}{2}$$
$$=\frac{|F_{E1}|(|F_{E1}|-1)}{2}+\frac{|F_{E2}|(|F_{E2}|-1)}{2}+\cdots\frac{|F_{En}|(|F_{En}|-1)}{2}$$
$$=\frac{|F_{E1}|^2+|F_{E2}|^2+\cdots|F_{En}|^2-(|F_{E1}|+|F_{E2}|+\cdots|F_{En}|)}{2}$$
$$=\frac{EF(T_s)-|F|}{2} \quad \text{where } EF(T_s)=\sum_{i=1}^{n}|F_{Ei}|^2$$

Edge Factor, EF, is introduced and defined above. Since |F| is a constant, the number of remaining edges is solely determined by the edge factor, which is the square sum of the cardinalities of all fault equivalence sets. Calculating the edge factor for a candidate test set is simple and can be done in the time of O(|F|) by a specific implementation described later. The proposed EF-based greedy algorithm is shown in Fig. 6.

Calculation of EFs dominates the computation complexity of the proposed algorithm and special cares need to be taken in the implementation details. Obtaining EF for a given Ts requires completing two tasks: 1) find which fault equivalence set each fault belongs to; 2) find the cardinality of each fault equivalence set. For the first task, an SID of a fault f_i with respect to T_S is a sequence defined as

$$SID_{f_i|T_s}=[w_1w_2w_3\ldots w_{|T_s|}]$$

where w_k indicates whether the k^{th} vector in T_S can detect f_i. Each fault has a bit recording whether it is detected by a specific test vector or not, so the length of SID depends on the size of T_S. Faults with the same SID belong to the same F_E because they can not be distinguished under the given T_S. Fig. 7 shows an example of how to get SIDs for a given T_S.

For the second task, a hash table of size M is constructed to count the number of faults in each fault equivalence set, where M is selected as a prime number slightly larger than |F|. The proposed algorithm scans all the faults one by one, calculates their SIDs under a given T_S, and then searches for their SIDs in the hash table. If there is a search hit, the counter of the associated SID is incremented by 1. This implies the current fault belongs to an existing fault equivalence class. Otherwise, the algorithm creates a new node in the hash table. This node is associated with the SID of the current fault and the corresponding counter is initialized to 1. Note that there is at most $\min(2^{|Ts|},|F|)$ nodes in the hash table. Typically, $\min(2^{|Ts|},|F|)$ is limited to |F| as T_S becomes large enough. Hence, a hash table with size M > |F| ensures the loading density < 1 all the time. Empirically, this implies on average 1.5 search is merely required to check whether an SID is in the hash table or not [11]. After the hash table is successfully constructed, every single node in the hash table represents a unique fault equivalence set and the associated counter indicates its cardinality. An example of the construction process of the hash table for T_S given in Fig. 7 is shown in Fig. 8. The last step is simply traversing the hash table and calculating the $EF(T_S)$. Both of these two steps, searching all SIDs within the hash table and calculating the EF, take about O(|F|) time.

```
Greedy_EF(F, T, N) {
  Ts ← ∅
  do
    for i ← 1 to |T|
      if ti ∉ Ts
        Yi ← Ts ∪ {ti}
        calculate EF(Yi)
    identify tj with the smallest EF
    Ts ← Ts ∪ {tj}
  while (EF(Ts) > |F| and |Ts| < N)
  return Ts
}
```

N is the given upper bound of the number of selected vectors
T_S is the set of selected test vectors
Y_i is a temporary test set

Fig. 6. The pseudo code of the proposed EF-based algorithm.

	f_1	f_2	f_3	f_4	f_5
$T_S=\{t_1\}$	0	0	0	1	1
$T_S=\{t_1,t_3\}$	01	01	00	10	11
$T_S=\{t_1,t_3,t_2\}$	010	011	001	101	111

Fig. 7. SIDs corresponding to growing selected test sets.

Note that neither the distinguishability table nor the transformed graph needs to be explicitly constructed in this algorithm. Instead, the edge factor guides the iterative test selection process throughout the algorithm. By utilizing the above implementation, it takes merely O(|F|) time to obtain the EF for a given test set. So given a upper bound of the number of selected test vectors, N, the time complexity of Greedy_EF is O(N*|T|*|F|). Note that N is set to the ideal $\lceil \log_2|F| \rceil$ in this paper for maximum possible dictionary size reduction. Therefore, the time complexity is an extremely low $O(|T|*|F|*\log_2|F|)$. Moreover, the memory requirement for Greedy_EF is also extremely low. The length of an SID is at most |T| bits and usually $O(\log_2|F|)$ bits in practical use. Hence the hash table requires only $O(|F|*\log_2|F|)$ space. Hence the space complexity of Greedy_EF is O(|T|*|F|) due to the construction of the initial pass-fail dictionary. We conclude that the proposed algorithm indeed possesses low time and space complexity.

V. Experimental Results

The proposed algorithm is implemented in C++ with the data structures described in Section IV. All experiments are

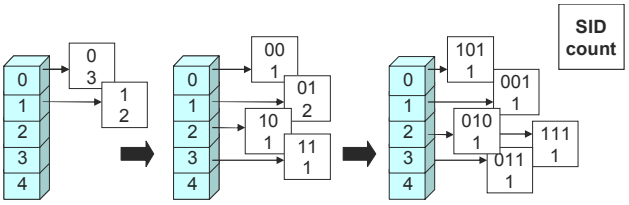

Fig. 8. Hash table facilitating the computation of EF.

TABLE I
Results on ISCAS'85 and'89 Benchmark Circuits

Circuit	|T|	|F|	$\lceil \log_2\|F\| \rceil$	DR before reduction	DR after reduction	DR in [6]	Size Reduction ratio	CPU (s)	Mem (MB)
c432	50	524	10	0.995877	0.989374	0.993403	0.800	0.03	<1
c499	53	758	10	0.999456	0.997121	0.997257	0.811	0.04	<1
c880	54	942	10	0.999621	0.996950	0.996970	0.815	0.06	<1
c1355	86	1574	11	0.999241	0.997302	0.997274	0.872	0.15	<1
c1908	120	1879	11	0.999403	0.997000	0.997063	0.908	0.26	<1
c2670	106	2747	12	0.997739	0.996207	0.996088	0.887	0.35	<1
c3540	150	3428	12	0.998169	0.996568	0.996694	0.920	0.64	<1
c5315	125	5350	13	0.999776	0.999268	0.999083	0.896	0.91	<1
c6288	30	7744	13	0.999815	0.999578	0.999555	0.567	0.27	<1
c7552	217	7550	13	0.999566	0.998992	0.998871	0.940	2.22	2
s9234	384	6927	13	0.995602	0.993234	--	0.966	3.56	2
s13207	460	9815	14	0.999591	0.998303	--	0.970	6.72	4
s15850	438	11725	14	0.998871	0.997933	--	0.968	7.69	6
s35932	68	39094	16	0.989591	0.989422	--	0.765	4.02	6
s38417	901	31180	15	0.999952	0.999587	--	0.983	41.69	28
s38584	654	36303	16	0.998275	0.997943	--	0.976	42.47	24

performed on a Pentium4 3.0 GHz platform.

First, we apply our method to ISCAS'85 and ISCAS'89 benchmark sets [12], [13]. The Atalanta test generator [14] and HOPE fault simulator [15] are used to generate the original fault dictionary for each circuit. In order to meet the industrial practice, the compaction techniques in Atalanta are turned on. To improve the diagnostic resolution, the fault dictionaries are preprocessed by XORing the responses of the test vectors as in [6]. Table I lists the experimental results of the benchmark circuits. The second and third columns denote the number of test vectors and faults, respectively. The size of the selected test set T_S is set to $\lceil \log_2|F| \rceil$ shown in the forth column in order to directly compare the diagnostic resolutions reported in [6]. The fifth and sixth columns denote the diagnostic resolutions before reduction and after reduction. The seventh column denotes the diagnostic resolutions revealed in [6]. The eighth column denotes the fault dictionary reduction ratio. The results show that our diagnostic resolutions are comparably good as those reported in [6] for ISCAS'85 benchmark circuits. The average size reduction ratio is up to 87.8%, while the average DR loss is only 0.00161, less than 0.2%. To sum up, the fault dictionary size reduction ratio is very significant with merely a slight loss in DR.

The ninth and tenth columns in Table I show the CPU time and memory usage of our proposed algorithm. The runtime is remarkably short. Even the largest circuit in the benchmark set can be processed within a minute. Along with the low time complexity, $O(|T|*|F|*\log_2|F|)$, the proposed method is capable of dealing with practical large industrial circuits. Besides the time complexity, the other obstacle preventing the distinguishability table to practical applications is its unacceptable large size. By using the edge factor instead, the proposed algorithm reduces the memory usage to about the size of the original fault dictionary. The largest circuit in Table I requires only 24MB throughout the whole reduction process. As discussed previously, without the explicit construction of the distinguishability table, the proposed algorithm performs impressively well with short runtime and low memory usage. Though we cannot directly compare our results to those of [6], in which no runtime or memory information is provided, the experimental results along with the complexity analysis presented in Section IV should be enough to ensure the superiority of the proposed algorithm in both aspects.

The circuits in Table I are not large enough to demonstrate the proposed algorithm's capability of handling large industrial circuits. Hence, for the second part of the experiment, the proposed algorithm is applied to several large randomly generated fault dictionaries. The distribution of '1' and '0' entries in the generated fault dictionaries is set to be equal to the average of all ISCAS'85 and '89 benchmark circuits. The results on those weighted random fault dictionaries are shown in Table II. The first and the second columns denote the number of test vectors and faults. The fourth column denotes the diagnostic resolution. Possibly due to the random characteristic of the dictionaries, the diagnostic resolutions are impressively high. The fifth and sixth columns show the runtime and memory usage, respectively. For most cases, the runtimes are kept within one hour. Even the largest case possessing million faults takes merely around an hour to complete the whole reduction process. The memory usage is also low in all cases; none of them exceeds 1GB. Though these randomly generated fault dictionaries are not derived from real circuits, the DR cannot serve more than a reference. However, applying on the large size dictionaries directly validate time and space efficiency of the proposed algorithm. It should be reasonable to claim that a modest computer nowadays can run the proposed algorithm to process million-gate large circuits.

The proposed method exhibits great efficiency in both runtime and memory usage as shown in Table I and II. It suggests that the proposed edge factor can be used as a good

guidance or cost function in more sophisticated algorithms which may further boost up the diagnostic resolution and thus allow smaller fault dictionary. For example, we can keep several best candidates in each test vector selection iteration of the algorithm to get a more globally optimized solution. Another extension is to use simulated annealing based algorithms with the edge factor as the cost function. The advantage of low memory complexity in our proposed method is still preserved in the above two kinds of algorithm. Although the details need more research efforts, we believe the proposed edge factor is promising to be used in other algorithms.

VI. Conclusions

A time- and space-efficient approach for size reduction of fault dictionary is proposed in this paper. Edge factor, being strictly proportional to the number of undistinguished fault pairs, is introduced as the guidance throughout the algorithm. With correct implementation techniques, the edge factor can be calculated directly from the original pass-fail fault dictionary and thus the time- and memory-consuming construction of the distinguishability table and further operations on that table are completely excluded. Experimental results over sets of benchmark circuits validate the effectiveness and efficiency of the proposed algorithm. Hence, the proposed algorithm is capable of providing a feasible solution of fault dictionary compaction for today's industrial million-gate circuits. Moreover, the edge factor is promising to be applied in more sophisticated algorithms for further research works.

Acknowledgment

This work was supported in part by the National Science council of Taiwan, R.O.C., under Grant NSC 94-2220-E009-041. The authors also want to thank the anonymous reviewers for their valuable comments.

TABLE II
Results on Large Randomly-Generated Fault Dictionaries

|T|	|F|	$\lceil \log_2 \|F\| \rceil$	DR	CPU (sec)	Mem (MB)
1000	50000	16	0.999986	95.970	48
1000	100000	17	0.999990	222.510	96
1000	500000	19	0.999996	1579.220	492
1000	1000000	20	0.999998	3733.180	972

References

[1] M. Abramovici, M.A. Breuer and A.D. Friedman, "Digital systems testing and testable design," Computer Science Press, 1990.

[2] F. Kocan and D.G. Saab, "Dynamic fault diagnosis on reconfigurable hardware," in *Proc. Of IEEE/ACM Design Automation Conference*, 1999, pp. 691-696.

[3] R. Li, J. H. Olson and D.L. Chester, "Dynamic fault detection and diagnosis using neural networks," in *IEEE International Symposium on Intelligent control*, 1990, vol.2, pp. 1169-1174.

[4] S. Venkataraman, I. Hartanto and W. Kent Fuchs, "Dynamic diagnosis of sequential circuits based on stuck-at faults," in *IEEE VLSI Test Symposium*, 1996, pp. 198-203.

[5] I. Pomeranz and S.M. Reddy, "On the generation of small dictionaries for fault location," In *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, 1992, pp. 272-279.

[6] B. Arslan and A. Orailoglu, "Fault dictionary size reduction through test response superposition," in *Proc. of IEEE International Conference on Computer Design,* 2002, pp. 480-485.

[7] P.G. Ryan, W.K. Fuchs and I. Pomeranz, "Fault dictionary compression and equivalence class computation for sequential circuits," in *Proc. of IEEE/ACM International Conference on Computer-Aided Design,* 1993, pp. 508-511.

[8] V. Boppana and W.K. Fuchs, "Fault dictionary compaction by output sequence removal," in *Proc. of IEEE/ACM International Conference. on, Computer-Aided Design,* 1994, pp. 576-579.

[9] V. Boppana, I. Hartanto, and W.K. Fuchs,, "Full fault dictionary storage based on lable tree encoding, " in *Proc. of IEEE VLSI Test Symposium,* 1996, pp. 174-179.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to algorithms," The MIT Press, 2003.

[11] E. Horowitz, S. Sahni, and D. Mehta, "Fundamentals of data structures in C++," W.H. Freeman and Company, 1995.

[12] F. Brglez and H. Fujiwara., "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," in *Proc. of International Symposium on Circuits and Systems*, June 1985.

[13] F. Brglez et al., "Combination Profiles of Sequential Benchmark Circuits", in *International Symposium on Circuits and Systems*, May 1989, pp. 1929-1934.

[14] H.K. Lee and D. S. Ha, " *On the Generation of Test Patterns for Combinational Circuits*," Technical Report 12-93, Department of Electrical Eng., Virginia Polytechnic Institute and State University.

[15] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator," in *Proc. of ACM/IEEE Design Automation Conference*, pages 336–340, 1992.