記憶體節省修復解之代表元件
Representatives of Economical Repair Solutions for Memories

梁新聰
Hsing-Chung Liang

Abstract

This paper introduces a novel procedure of identifying better representatives of faulty cells in a memory map to help judge unrepairability and provide economic repair recommendation. These representative faulty cells, called leading elements (*LE*), are classified into four primary types based on their characteristics. Three specific pairs of initially identified *LE* are extracted for further operations, which are replacing certain LE with other better representatives and assigning the cross point faults between two certain LE as new LE. All steps of the procedure are analyzed in sequence with verification, clearly indicating that the identified LE represent both the more exact thresholds for judging unrepairability and usually the most economic repair solutions. Experiments on many example maps show that the procedure can be fast in searching 7% more LE and be applicable to accumulate data for redundancy planning afterwards.
Keywords: repair solution, unrepairability, memory map

摘要

本論文提出一個新的程序，選出記憶體之錯誤元件中，以何者為較佳代表，做為輔助判斷記憶體是否不可修復，以及提供最經濟之修復建議。這些代表元件依其特性分類為四種主要型式；起初判斷之代表元件中，進一步處理其中三種型式，包括將其中一些代表元件，以其他較佳之代表元件取代之，或是將兩個代表元件之交錯元件，也設為代表元件之一員。所有這些程序之步驟，經循序分析驗證，清楚指出所定之代表元件，同時表示著較精確的不可修復判定標準，以及經常為最經濟之修復解。執行此程序於許多實驗例子，比較結果顯示其速度非常快，可比原方法多找出百分之七的代表元件，因此其可應用於收集資料，做為多餘元件設計準備之參考。
關鍵詞：修復解，不可修復，記憶體圖

I. Introduction

As required for many designs like system-on-a-chip (SoC) or system-in-package (SIP), memories usually consist of the very large part of a complete chip or system. Their yields are very important, therefore demanding built-in redundancy to replace detected faulty sections. Redundancy for memory maps can be spare lines like ***spare rows*** (***SR***) and/or ***spare columns*** (***SC***). Deciding how to use spare lines for replacing faulty cells is a NP-complete problem [1] and needs heuristics for achieving solutions. Many methods, called reconfiguration algorithms, have been proposed to help search the repair solutions and are roughly classified into the following categories:

1. Use bipartite graphs to model the problem and then choose appropriate nodes to cover all edges [1] - [7].
2. Use branch-and-bound methods to store the repair selections in a binary-tree or ternary-tree and search the repair solutions accordingly [7] - [10].
3. Use cost functions to decide the repair solutions [7] [10]-[13].

In addition to single stuck-at faults, other faults were also targeted, e.g. sparse faults [14], cluster faults [15]-[17], or coupling faults [3]. It was suggested to use a neural network to model the problem of searching repair solutions [18]. Identifying unrepairable maps with strategies or checking bounds is also useful in speeding up the complete process of repair decisions [8][19][20].

中原大學電子系
* Corresponding author. E-mail: hcliang@cycu.edu.tw
Department of Electronic Engineering, Chung Yung Christian University, 200 Chung Pei Road, Chung Li, Tao Yuan 320, Taiwan, R.O.C.
Manuscript received 16 May 2006; revised 2 October 2006; accepted 30 October 2006

Strategies or checking bounds can be organized in sequence to distinguish unrepairable maps from repairable ones. Although some are too loose to check high-yield memories, the more exact ones were proved to have a very helpful speed-up to methods of searching repair solutions [20]. In addition, the ***leading elements*** (***LE***) defined in [19] were extended to more types, inducing a more exact threshold to judge unrepairability in [20]. Given a faulty map, if the induced threshold is larger than the given redundancy, the map is definitely unrepairable. Otherwise, it needs further checking for being repairable or not. Certain identified *LE* seems to directly provide economic repair solutions. However, as discussed by the example in Section II, replacing some *LE* searched by the procedure of [20] may generate new *LE* and induce unpredictable iterations for judging unrepairability. In addition, if not easy in deciding repair solutions or judging unrepairability, it needs another searching procedure to decide final solutions. In this paper, we propose a procedure to search *LE* representatives in more exact way to help judge unrepairability and simultaneously plan for economic repair solutions. Section III analyzes the characteristics of faulty cells to introduce the improved procedure. Section IV verifies the procedure with experimental results. Section V gives the conclusion.

II. PRELIMINARIES

We use some definitions and notations from [19] & [20]. A memory map is assumed to have *SR* and *SC*, prepared for repairing faulty cells. The first *LE* in a memory map defined in [19] is the first faulty cell on the first faulty row. Ignoring the faulty cells on the row & column of the first *LE*, we can identify the next *LE* in the similar way. It's obvious that no two *LE* are allowed to be on the same row or column. A faulty cell without other faulty cells on the same row & column is called a ***single fault*** (***SF***), which must be a *LE*. Other faulty cells on the row or column of a *LE* are called the ***row*** or ***column complements*** of the *LE*, denoted as ***RC*** & ***CC***, respectively. Here we extend this definition so that a faulty cell is a *RC* (*CC*) of another one, not restricted on *LE*, if they are on the same row (column). As in [20], a *RC* (*CC*) of a *LE* is ***covered*** by another *LE* if they are on the same column (row), called the ***cross point fault*** (***CPF***) of the two *LE*. If a fault is only covered by one *LE*, it's denoted as a **non-*CPF*** of the *LE*. If a *LE* has at least one *RC* uncovered by other *LE*, i.e. non-*CPF*, it's marked with ***UCR***. Similarly, we mark a *LE* with ***UCC*** if it has at least one *CC* of non-*CPF*. A *LE* is called an ***AC*** if its *RC* and *CC* are all *CPF* with other *LE*. If a *LE* has non-*CPF* in its *RC* & *CC*, i.e. it has both the characteristics of *UCR* & *UCC*, it's denoted as a ***BUC***. A *BUC* can be substituted by two additional *LE* like the procedure *UCR_UCC* in [20]. After tackling all *BUC*, we can take the number of total *LE* as a minimum *threshold* for the number of prepared spare lines in order to identify the unrepairable memories as early as possible. Fig. 1 gives an example map of four *LE*, obtained by iteratively assigning the first-met faulty cell as the *LE* [20]. The faulty cell on the fourth row and first column is denoted as (4,1). The fault (4,3) is a *UCC* because (6,3) is a non-*CPF*; it can be denoted as *UCC*(4,3). The fault (2,6) is the *CPF* of *UCR*(2,1) & *AC*(1,6). *UCC*(4,3) & *UCC*(5,8) have no common *CPF*.

Obviously, a *UCR* is more economic to be repaired with a *SR* than with a *SC* because it needs only one *SR* to repair all faulty cells on its row. If instead repairing it with a *SC*, one of its non-*CPF* becomes a new *LE* and requires further redundant lines for repairing. Similarly, a *UCC* is more economically repaired with a *SC* than with a *SR*. Fig. 1 has four *LE* and therefore requires at least four spare lines for repairing. In fact, it needs one more line because the cell (4,1) will become a new *LE* after substituting a *SR* for *UCR*(2,1) and a *SC* for *UCC*(4,3), respectively. It seems that the original searching procedure is incomplete for some example maps. In Section III, a procedure to search more exact *LE* is proposed with required analysis and proof. In addition to providing better thresholds for judging unrepairability, the obtained *LE* can recommend economic repair solutions if being accumulated in statistic.

III. PROCEDURE OF DECIDING *LE*

1. First four steps

The first four steps are kernel of the procedure. They try to choose the suitable faulty cells as the *LE* representatives in a memory map so that the later steps need as less work as possible in reallocation. Obviously, a faulty row (column) with at least one faulty cell having no *CC* (*RC*) is more economically repaired with a *SR* (*SC*). The first step therefore searches the faulty rows (columns) having at least one faulty cell without *CC* (*RC*) and sets these special faulty cells as the *LE* representatives. It temporarily ignores the *LE* and their *RC* & *CC*, and checks the remaining faults in the same way. If such types of faulty rows & columns occupy the most part of a memory map, this step has almost dealt with all faulty cells and provided the repair solutions if the spare lines are enough. However, there may be still cases like clustered faults or scratch damage,

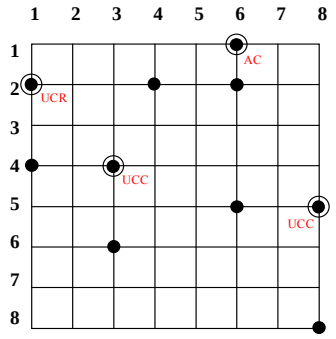

Fig. 1 An example map in [20]

causing that each remaining fault after Step 1 has at least one *RC* and one *CC*. Step 2 therefore tries to check the remaining cells to select suitable *LE*, e.g. alternately searching the rows & columns with less number of faulty cells. Given such a row (column), it sets the uncovered fault with the least number of *CC* (*RC*) as the *LE* representative. With the first two steps, the procedure makes every fault except *LE* be covered by at least one *LE*. Fig. 2 shows the resultant map for the example of Fig. 1, which can provide five *LE* for more exact threshold of prepared spare lines.

After the first two steps, some *LE* are *BUC* as they have at least one *RC* and one *CC* being not covered by other *LE*. For a *BUC*, it needs at least one *SR* and one *SC* to replace the faulty cells on its row and column, respectively. The procedure therefore checks the existence of *BUC* in Step 3 and substitutes two suitable faulty cells as *LE* representatives for a *BUC* until no *BUC* exists. The agents are two of its non-*CPF*, particularly one *RC* with the least number of *CC* and one *CC* with the least number of *RC*. The first three steps try to make every *LE* be with the property of having the least number of *CC* among the faulty cells on its row and the least number of *RC* among the faulty cells on its column. This makes the determined *LE* be almost the final repair solutions with economic consideration. Step 4 temporarily marks the *LE* with *SF*, *AC*, *UCR*, or *UCC* according to their characteristics, as shown by Fig. 3 for the example of Fig. 2. The LE make the required spare lines increase to five as compared to that given in Fig. 1. To ensure identifying more precise *LE*, we will analyze *CPF* in the following subsections to introduce the subsequent steps of the improved procedure.

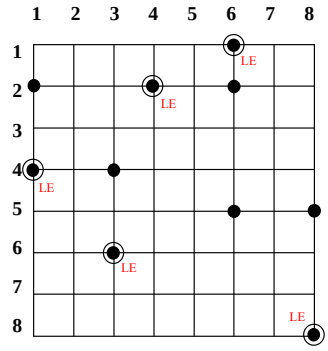

Fig. 2 *LE* of Fig. 1 obtained by the procedure

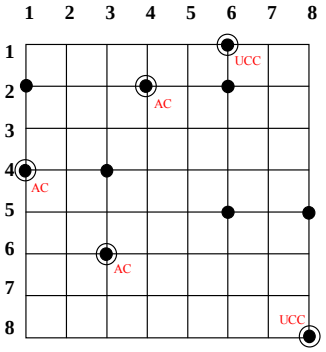

Fig. 3 Map of Fig. 2 obtained after Step 3

2. Nine types of *CPF*

As defined previously, a *CPF* is a non-*LE* covered by two *LE*, i.e. row LE (*RLE*) and column LE (*CLE*) on the same row and column of the *CPF*, respectively. After Step 4, *LE* can be of four types: *SF*, *AC*, *UCC*, & *UCR*. Since a *SF* can't be a *LE* of a *CPF*, there can be nine possible *LE* combinations around a *CPF*, as shown in Fig. 4. Certain types of *CPF* have the feature that their *LE* do not affect each other in deciding repair solutions, e.g. type-1 *CPF* in Fig. 4(a), which is a *CPF* with both *RLE* & *CLE* being *UCC*. Since the first three steps search a faulty cell with the least *RC* as the *UCC* on a column, each non-*CPF CC* of *UCC*(1,1) in Fig. 4(a) has at least one *RC*. In addition, for each non-*CPF CC* of *UCC*(1,1), all its *RC* have *CLE* of *UC C*, instead of *AC* or *UCR*, because the *CC*'s row has no *LE*. The columns of *UCC*(1,1) & *UCC*(2,2) are more economic to be repaired with *SC* than with *SR*. In addition, repairing *UCC*(1,1) with a *SC* does not affect the suitable repair solution for *UCC*(2,2), and vice versa. Similarly, the *UCR*(2,2) in Fig. 4(b) can occur only when each of its non-*CPF RC* has at least one *CC* and all the *RLE* of these *CC* must be *UCR*. Repairing one *UCR* with a *SR* does not affect the other; furthermore, this repair selection is also most economic if the spare lines are enough. For convenience, in the following we will use ***least_repair*** to indicate

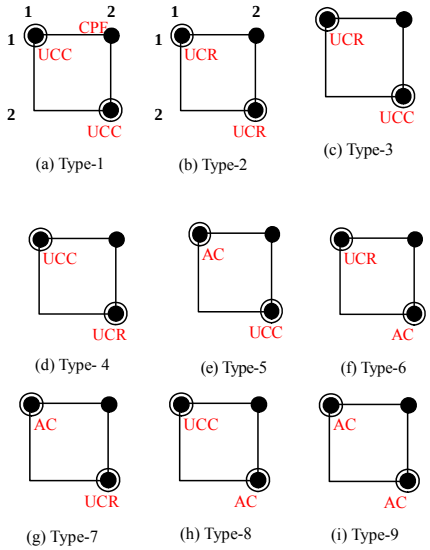

Fig. 4 Nine types of *CPF*

the method of repairing faulty cells on a *UCR* row with a *SR* and those on a *UCC* column with a *SC*.

A type-3 *CPF*, as shown in Fig. 4(c), has the similar condition and is most economic with *least_repair*. For a *CPF* of type-5 or type-6, all *RC* and *CC* of the *AC* are still *CPF* after applying the *least_repair*. The *AC* will become

a *SF* only if it has no other *CC* & *RC* except the *CPF* in these sub-figures. The following lemma summarizes the above analysis and suggests us needing not to deal with the mentioned types of *CPF* in the later steps of the procedure.

Lemma 1: For a *CPF* of type 1, 2, 3, 5, or 6, applying *least_repair* on one of its *LE* does not change the nature of another *LE* except making it become a *SF* (for type-5 & type-6 *CPF*).

3. *CPF* of type 7 & 8

Step 5 of the procedure is dealing with *CPF* of type 7 & 8. After running the first four steps, a type-7 *CPF* like Fig. 4(g) can occur because of the following three conditions: (1) each *RC* of *UCR*(2,2) has at least one *CC*, (2) *UCR*(2,2) has at least one non-*CPF*, (3) all the *CC* of *UCR*(2,2) and all the *RC* & *CC* of *AC*(1,1) are *CPF* with other *LE*. Obviously, the *UCR*(2,2) can be substituted by one of its non-*CPF RC*, which turns *CPF*(1,2) into a non-*CPF*. The agent for *UCR*(2,2) is still an *UCR*, but *AC*(1,1) becomes an *UCR*. A type-8 *CPF* has the similar feature. The following lemma is consequently obtained for processing the *CPF* of type 7 & 8 in the procedure.

Lemma 2: For a type-7 *CPF*, applying *least_repair* for economic repairing makes the correlative *AC* become an *UCR*. Similarly, for a type-8 *CPF*, applying *least_repair* makes the correlative *AC* become an *UCC*.

For a type-7 *CPF*, the remarking of *AC* to *UCR* may cause its *RC* to become a *CPF* of type 2, 3, or 6; its *CC* to become a *CPF* of type 2, 4, or 7. As mentioned in Lemma 1, the *CPF* of type 2, 3, & 6 need not further processing in the procedure. The new generated *CPF* of type 7 will be processed in the same way in this step until no such *CPF* exists. The procedure deals with type-8 *CPF* in the similar way. The *AC* of the *CPF* is changed to an *UCC*. Its *RC* and *CC* may become *CPF* of type 1, 4, 8 and 1, 3, 5, respectively. The remaining types of *CPF* after Step 5 are those of type 4 & 9, as discussed in the following.

4. *CPF* of type 4

For a *CPF* of type-4 in Fig. 4(d), its *UCC* has a specific characteristic similar to that of *UCC*(1,1) of Fig. 4(a). In other words, for such type of *CPF* to occur, each non-*CPF CC* of *UCC*(1,1) must have at least one *RC* and all its *RC* are covered by other *UCC* in the column direction. Similarly, each non-*CPF RC* of *UCR*(2,2) must have at least one *CC* and all its *CC* are covered by other *UCR* from the row direction. If applying the *least_repair* to Fig. 4(d), *CPF*(1,2) with its other *RC* and *CC* not shown in the figure will need additional *SR* and/or *SC* for repairing. We can make *CPF*(1,2) become a new *LE*, being of a *SF*, *UCR*, *UCC*, or *BUC*, decided by its characteristic.

5. *CPF* of type 9

After the previous steps, the remaining *CPF* are of type 1, 2, 3, 5, 6, and 9. The *UCR*, *UCC*, and their related *CPF* will be replaced with spare lines after applying the *least_repair*. Consequently, the remaining target faults are *SF*, *AC*, and type-9 *CPF*. *SF* can be repaired with either *SR* or *SC*. The *AC* and their related *CPF* can be classified into *independent* groups. Two groups G_1 and G_2 are independent if each *AC* in G_1 has no common *CPF* with any *AC* in G_2, and vice versa. For a group with *n AC*, all its *AC* and related *CPF* can be repaired with *n SR*, or *n SC*, or some combination of *SR* & *SC*. Instead of deciding the repair solutions for each independent group by using algorithms like LECA [10] or FCECA [20], we can alternately assign *SR* & *SC* for each *AC*. *SF* can be considered at last to obtain a balanced or acceptable usage of redundancy. This heuristic can fast provide the repair suggestions that are very close to the most economic solutions.

6. The Improved Procedure

The proposed procedure to determine *LE* for unrepairability judging and economic repair solution is summarized with the following steps.

Step 1. Check each row and column with faulty cells. If a faulty row (column) has at least one faulty cell without *CC* (*RC*), select one as a *LE* representative. Each faulty row or column has at most one *LE*. After checking all the faulty cells, we assume that the *LE* and their related *CC* (*RC*) are replaced with spare lines. The other faults are checked similarly until each remaining fault has at least one *RC* and one *CC*.

Step 2. Check if a faulty line has no *LE* but still has faulty cells not covered by other *LE*, i.e. uncovered faulty cells. Select one uncovered faulty cell as the *LE* representative. Continue this step until each faulty cell is covered by at least one *LE*.

Step 3. For each *LE* being a *BUC*, select one of its *RC* and one of its *CC* to be the *LE* substitutes for the *BUC*. The chosen *RC* (*CC*) should be a non-*CPF* and has the least number of *CC* (*RC*). Continue this step until no *BUC* exists.

Step 4. For each *LE*, classify it as *SF*, *UCR*, *UCC*, or *AC* in reference to its characteristic.

Step 5. Check if there are *CPF* of type 7 & 8. For a type-7 *CPF*, move its *UCR* to another representative and rename its *AC* to *UCR*. For a type-8 *CPF*, move its *UCC* to another representative and rename its *AC* to *UCC*. Process the two types of *CPF* alternately or with some suitable sequence until no such *CPF* exists in the memory map.

Step 5. Check if there are *CPF* of type 4. For a type-4 *CPF*, move its *UCR* & *UCC* to another representatives respectively and mark the *CPF* as a new *LE*.

Step 6. Classify the type-9 *CPF* and their *AC* into independent groups. Consider these groups and all *SF* to determine the utilization plan of remaining spare lines.

IV. IMPLEMENTATION AND EXPERIMENTS

Faulty cells on a memory map are roughly classified into sparse faults, long faulty lines, or clustered faults, in which the last one is primarily considered during the search of repair solutions. Let B(n,p) represent a $n \times n$ block of memory cells with p faulty cells. B(3,3), B(4,5) & B(5,8)

are examples in which about 30% cells are faulty in a block. They can have 84, 4368, & more than 1.08 million possible examples, respectively, if each faulty cell has its own address. The following experiments tackle suitable sizes and randomly select 10000 examples for each target number of faults in a block to obtain the average results. The same faulty example that has been chosen before is not considered again during selection. These example blocks are run on a PC with a 1.5GHz CPU & 256 MB RAM.

Table 1 shows the average *LE* distribution for a variety of block sizes. The second column gives the numbers of faults in one block chosen for the cases with largest summation of occurring percentages of *UCR* and *UCC*. For 5×5 & 4×4 blocks, the largest percentages of *UCR* & *UCC* occur at the condition that one-third of total cells are faulty, while those for 6×6, 7×7, and 8×8 blocks occur when one fourth of total cells are faulty.

Table 2 shows the average distribution of *LE* for 1Mb memory maps with 100 randomly distributed faults, i.e. with 0.01% occurring probability. The first column represents a variety of distributing regions for these faults. The second & third columns give the resultant mean and standard deviation of *LE* from 10000 examples for each distribution size. For cluster regions close to 20×20, most *LE* are *AC*. For larger regions, they behave more like *UCR* & *UCC*. When the 100 faults are sparse in regions larger than 200×200, most *LE* will be *SF* to dominate the repair solutions. We can summarize from this table that:

1. If the 100 faults are clustered in a very small region, most *LE* are *AC* and these faulty cells can be replaced with less spare lines than those distributed in larger areas.
2. If these faults are clustered in a region ranging from 30x30 to 100x100, most *LE* are *UCR* & *UCC* for repair decision.
3. If these faults are distributed in an area larger than 100x100, most *LE* behave as *SF* and need many spare lines for replacement, which is not economical at all. It may be more economical by replacing the faulty cells with spare blocks instead.

With the comparison of Fig. 1 and Fig. 2, the improved procedure is predicted to identify more *LE* than iteratively choosing the first-met faulty cell as *LE*, which was proposed in [19]. For the blocks with sizes varying from 20×20 to 90×90 for the 100 faults to be randomly distributed, Fig. 5 shows the additional *LE* percentages achieved by the complete procedure and by the first four steps, respectively. For the example maps with 150 faults, the improved procedure also achieves about 7.3% more LE than the first-met method. The most improvement occurs on 50×50 clustering blocks instead. Comparing to the previous procedure, whose searching time was about 1 ms in average per map [20], the improved procedure takes only 15 μs additionally to identify more *LE* representatives. Obviously, these *LE* provide more exact thresholds to judge unrepairability and usually consist of very economic repair solutions because most of them are *UCR* & *UCC* for general examples.

Table 1 Average distribution of *LE* for various blocks with the largest % of (*UCR*+*UCC*)

block	#faults	*UCR*	*UCC*	*AC*	*SF*
10*10	20	38.5%	44.3%	13.2%	4.0%
9*9	17	40.9%	40.6%	14.3%	4.2%
8*8	16	40.3%	39.9%	16.7%	3.1%
7*7	13	40.0%	39.9%	16.1%	4.0%
6*6	9	39.1%	39.4%	12.4%	9.1%
5*5	8	38.4%	38.4%	19.3%	3.9%
4*4	5	38.4%	38.5%	6.9%	16.2%

Table 2 Average *LE* for various distributing regions of 100 faults on a 1Mb map

distribution size	*LE*	σ	*UCR*	*UCC*	*AC*	*SF*
20×20	19.9	0.32	1.4	1.3	17.2	0.0
30×30	28.4	1.02	11.2	11.2	6.0	0.1
40×40	34.7	1.38	15.6	15.6	3.0	0.6
50×50	39.5	1.71	17.3	17.4	3.1	1.8
60×60	43.5	1.95	18.3	18.3	3.4	3.5
70×70	46.9	2.14	18.8	18.8	3.5	5.8
80×80	49.7	2.30	19.1	19.1	3.3	8.2
90×90	52.4	2.42	19.1	19.3	3.0	11.0
100×100	54.8	2.58	19.1	19.2	2.7	13.7
200×200	69.3	3.07	15.7	15.7	0.8	37.1
300×300	76.7	3.19	12.4	12.4	0.3	51.7
400×400	81.2	3.08	10.1	10.0	0.1	61.0
500×500	84.2	3.01	8.4	8.4	0.1	67.3
1000×1000	91.2	2.53	4.6	4.6	0.0	82.0

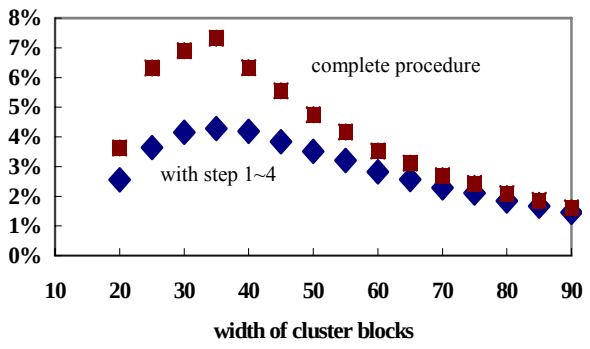

Fig. 5 Additional *LE* percentages achieved by the complete improved procedure and by the first four steps as compared to iteratively choosing the first-met faulty cell as *LE* [19]

V. CONCLUSION

In conclusion, we propose a procedure of searching representatives of faulty cells in memory maps to achieve better ones for deciding unrepairability and economic repair solutions. Faulty cells are classified into *LE*, *CPF*, and non-*CPF*, in which *LE* are the representatives of determining repair solutions. The *LE* can be further classified into four primary types, in which *UCR* & *UCC* usually consist of most *LE* and can be used directly to decide the most economic repair solutions. *SF* & *AC* can assist to balance the utilization of remaining spare rows & columns. Iteratively assigning the first-met faulty cell as *LE* done by the previous procedure is clearly incomplete to search all *LE*. A better procedure is consequently introduced to identify more exact *LE*, which can provide better thresholds for judging unrepairability with a little more time than the original one. Simultaneously, it can directly suggest repair solutions that are very close to the least required redundancy, which can also be statistically accumulated for application to spares planning of memory maps.

ACKNOWLEDGEMENT

The author is very grateful to the reviewers for their comments and suggestions to improve the paper.

REFERENCES

[1] S.-Y. Kuo and W. K. Fuchs, "Efficient spare allocation for reconfigurable arrays," *IEEE Design & Test of Computers*, pp. 24-31, Feb. 1987.

[2] N. Hasan and C. L. Liu, "Minimum fault coverage in reconfigurable arrays," *Proc. of IEEE International Symposium on Fault-Tolerant Computing*, pp. 348-353, 1988.

[3] M.-F. Chang, W. K. Fuchs and J. H. Patel, "Diagnosis and repair of memory with coupling faults," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 493-500, April 1989.

[4] R. W. Haddad, A. T. Dahbura and A. B. Sharma, "Increased throughput for the testing and repair of RAM's with redundancy," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 154-166, Feb. 1991.

[5] R. Libeskind-Hadas and C. L. Liu, "Fast search algorithms for reconfiguration problems," *Proc. of IEEE International Workshop on Defect and Fault Tolerance on VLSI Systems*, pp. 260-273, 1991.

[6] C. P. Low and H. W. Leong, "Minimum fault coverage in memory arrays: a fast algorithm and probability analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 6, pp. 681-690, June 1996.

[7] W. Che and I. Koren, "Fault spectrum analysis for fast spare allocation in reconfigurable arrays," *Proceedings of IEEE International Workshop on Defect and Fault Tolerance on VLSI Systems*, pp. 60-69, 1992.

[8] V. G. Hemmady and S. M. Reddy, "On the repair of redundant RAMs," *Proc. of the 26th ACM/IEEE Design Automation Conference*, pp. 710-713, paper 41.2, 1989.

[9] Y.-N. Shen and F. Lombardi, "Repairability/Unrepairability detection techniques for yield enhancement of VLSI memories with redundancy," *Proceedings of IEEE-Computers and Digital Techniques*, vol. 137, no. 2, pp. 133-136, March 1990.

[10] W. K. Huang, Y.-N. Shen and F. Lombardi, "New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 3, pp. 323-328, March 1990.

[11] S.-C. Fang, S.-J. Chen and S. L. Lee, "Novel graph-based algorithms for reconfigurable arrays," *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 2866-2869, 1991.

[12] F. Lombardi and D. Sciuto, "Repair of redundant memories by reduced covering," *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 205- 208, 1988.

[13] W.-K. Huang and F. Lombardi, "Minimizing the cost of repairing WSI memories," *Proc. of IEEE International Conference on Wafer Scale Integration*, pp. 183-192, 1989.

[14] C. P. Low and H. W. Leong, "A new class of efficient algorithms for reconfiguration of memory arrays," *IEEE Transactions on Computers*, vol. 45, no. 5, pp. 614-618, May 1996.

[15] B. Vinnakota and J. Andrews, "Repair of RAMs with clustered faults," *Proc. of IEEE International Conference on Computer Design*, pp. 582-585, 1992.

[16] D. M. Blough and A. Pelc, "A clustered failure model for the memory array reconfiguration problem," *IEEE Transactions on Computers*, vol. 42, no. 5, pp. 518-528, 1993.

[17] D. M. Blough, "Performance evaluation of a reconfiguration-algorithm for memory arrays containing clustered faults," *IEEE Transactions on Reliability*, vol. 45, no. 2, pp. 274-284, June 1996.

[18] N. Funabiki and Y. Takefuji, "A parallel algorithm for allocation of space cells on memory chips," *IEEE Transactions on Reliability*, vol. 40, no. 3, pp. 338-346, Aug. 1991.

[19] C.-L. Wey and F. Lombardi, "On the repair of redundant RAM's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 1, CAD-6, no. 2, pp. 222-231, March 1987.

[20] H.-C. Liang, W.-C. Ho and M.-C. Cheng, "Identify unrepairability to speed up spare allocation for repairing memories," *IEEE Transactions on Reliability*, vol. 54, no. 2, pp. 358-365, June 2005.