Efficient Repair Analysis Algorithm Exploration for Memory With Redundancy and In-Memory ECC

Minjie Lv, Hongbin Sun, *Senior Member, IEEE*, Jingmin Xin, *Senior Member, IEEE*, and Nanning Zheng, *Fellow, IEEE*

Abstract—In-memory error correction code (ECC) is a promising technique to improve the yield and reliability of high density memory design. However, the use of in-memory ECC poses a new problem to memory repair analysis algorithm, which has not been explored before. This article first makes a quantitative evaluation and demonstrates that the straightforward algorithms for memory with redundancy and in-memory ECC have serious deficiency on either repair rate or repair analysis speed. Accordingly, an optimal repair analysis algorithm that leverages preprocessing/filter algorithms, hybrid search tree, and depth-first search strategy is proposed to achieve low computational complexity and optimal repair rate in the meantime. In addition, a heuristic repair analysis algorithm that uses a greedy strategy is proposed to efficiently find repair solutions. Experimental results demonstrate that the proposed optimal repair analysis algorithm can achieve optimal repair rate and increase the repair analysis speed by up to $10^5\times$ compared with the straightforward exhaustive search algorithm. The proposed heuristic repair analysis algorithm is approximately 28 percent faster than the proposed optimal algorithm, at the expense of 5.8 percent repair rate loss.

Index Terms—Memory repair, repair analysis algorithm, in-memory ECC, yield, reliability

1 Introduction

Process scaling has steadily improved the density, performance, cost and energy efficiency of memory products. However, with the continuous scaling down of technology, serious challenges have been posed to memory design, especially for Dynamic Random Access Memory (DRAM). First, smaller DRAM cells and peripheral circuitry are more vulnerable to manufacturing variations and imperfections [1]. This causes various permanent defects, such as failures of entire row or column (cluster defects) and single-cell defects (SCDs), leading to the yield loss of DRAM products. In particular, the occurrence of SCDs drastically increases with process scaling down, making them the primary source of device failures [2]. Meanwhile, DRAM devices are more susceptible to soft errors induced by cosmic rays [3] and random telegraph noise [4], with the decrease in DRAM cell size and operating voltage. The increase in soft errors also seriously threatens system reliability. Therefore, the yield and reliability challenges have become the major obstacles that prevent the continuous scaling down of DRAM.

Redundancy repair [5], [6], [7], [8] and error correction code (ECC) are two traditional techniques used to improve the manufacturing yield and reliability of memory system, respectively. In order to improve the manufacturing yield, each DRAM array is equipped with several spare rows and columns to fix the permanent defects identified during memory test [5], [6], [9]. By replacing defective rows or columns with spare ones, redundancy repair technique is able to efficiently improve DRAM manufacturing yield for decades. During redundancy repair, repair analysis algorithm [10], [11], [12], [13] is used to provide an effective solution for allocating spare rows and columns. Besides, ECC is usually employed to protect memory data against soft error. Conventionally, ECC is utilized in DRAM at the rank level, where dual inline memory module with ECC (ECC-DIMM) implements single-error-correction and double-error-detection (SECDED) codes by providing an extra chip to store ECC check bits.

As SCDs have taken a large percentage of on-die defects with the scaling down of DRAM cells, it becomes inefficient to merely rely on redundancy repair for high yield. The estimated result in [2] shows that the bit error rate (BER) of SCDs in DRAM is expected to rapidly increase to around 10^{-4}, when DRAM process technology scales down to 1X and 1Y nodes. Moreover, advanced three-dimensional (3D) integration technology may further increase BER in future DRAM chips. It has been demonstrated that the overhead of traditional redundancy repair technique is unaffordable in the case of high BER of SCDs [2]. Besides, due to the increase of soft error rate, more efficient error tolerating techniques should be explored for next generation DRAM [2], [14], [15]. Therefore, DRAM researchers and manufacturers explore to place ECC inside DRAM dies, which is called in-memory ECC [16], [17], [18]. The possibilities and challenges

• *Minjie Lv is with the School of Cyber Engineering, Xidian University, Xi'an, Shaanxi 710126, China. E-mail: lv.minjie@stu.xjtu.edu.cn.*
• *Honbin Sun, Jingmin Xin, and Nanning Zheng are with the College of Artificial Intelligence, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China. E-mail: {hsun, jxin, nnzheng}@mail.xjtu.edu.cn.*

Manuscript received 26 Aug. 2019; revised 22 Mar. 2020; accepted 2 May 2020. Date of publication 22 May 2020; date of current version 7 Apr. 2021. (Corresponding author: Hongbin Sun.) Recommended for acceptance by O. Mutlu. Digital Object Identifier no. 10.1109/TC.2020.2996747

to implement in-memory ECC on DDR4 SDRAM devices are examined in [18]. Several DRAM chips and design prototypes with in-memory ECC have been reported [19], [20], [21], [22], [23]. There are also several works that explore the architectures of in-memory ECC either for yield improvement [2], [24] or reliability enhancement [25], [26].

The use of in-memory ECC poses a new problem: what is the efficient repair analysis algorithm for memory protected by both redundancy and in-memory ECC. In the case that in-memory ECC is employed for yield improvement, an optimal repair analysis algorithm taking these two techniques into consideration can achieve high yield. Even if in-memory ECC is employed mainly for reliability, a more efficient repair scheme can be achieved with small reliability degradation by borrowing a small amount of error correction capabilities provided by in-memory ECC. Although the repair analysis algorithm for memory with redundancy and in-memory ECC is a fundamental technique for future memory design, it has been rarely investigated in the research community. To the best of our knowledge, no previous study has been conducted on this technique.

This paper aims to make the first step towards exploring the efficient repair analysis algorithms for memory equipped with both redundancy and in-memory ECC. We first evaluate four straightforward algorithms through experiments and demonstrate that the simple use of conventional or slightly modified conventional repair analysis algorithms is inadequate for memory with redundancy and in-memory ECC, especially for high defect density scenarios. Although exhaustive search can achieve optimal repair rate, its repair analysis speed is extremely slow and unacceptable. Therefore, we propose an efficient optimal repair analysis algorithm, that uses preprocessing/filter algorithms, hybrid search tree (HST) structure and depth-first search (DFS) strategy to significantly reduce the computational complexity of exhaustive search and speed up repair analysis. Moreover, we further propose a heuristic repair analysis algorithm for memory with redundancy and in-memory ECC based on the observation during optimal algorithm exploration. The proposed heuristic algorithm essentially uses a greedy strategy to efficiently find the repair solution. Experimental results demonstrate that both of the proposed optimal and heuristic repair analysis algorithms can speed up the original straightforward exhaustive search algorithm by up to $10^5\times$ when the defect density is high. The proposed optimal algorithm can achieve the optimal repair rate, while the proposed heuristic algorithm is around 28 percent faster at the penalty of 5.8 percent repair rate loss. We believe that the exploration in this paper will encourage other researchers to exploit more efficient solutions for memory with redundancy and in-memory ECC.

2 Background

2.1 Memory Protected by Redundancy and In-Memory ECC

In current large-capacity DRAM designs, each memory die generally consists of several memory banks, where each bank is an independent array that has its own address and data buses and can be concurrently accessed. Modern DRAM designs usually use hierarchical design technique to improve the performance of high-density memory. Thus, DRAM bank is organized as a hierarchical structure, where each memory bank is composed of several blocks and each block is composed of dozens of subarrays. Fig. 1 illustrates the diagram of memory subarray protected by redundancy and in-memory ECC. There are usually hundreds or even thousands of rows (wordlines) and columns (bitlines) in a subarray. Redundancies are equipped to each subarray as spare rows and columns. These spare rows and columns can be used to repair defects by replacing the defective rows or columns through programming the one-time programmable laser fuses or electrical fuses associated to each subarray. When integrating in-memory ECC, a memory row can comprise dozens of ECC codewords, and each codeword consists of data bits and the corresponding check bits. The memory should integrate ECC logic, while DRAM external interface remains mostly unchanged. The embedded ECC logic encodes data written to DRAM to generate check bits and checks or corrects the data read out from DRAM array before transmitting them to the memory channel. For a large DRAM array with wide data bus (e.g., 16-bit data width), the data bus may be routed to several subarrays that can be concurrently addressed and accessed [27]. These subarrays share the same address bus and each of them provides several data bits (4, 2 or 1) to form the entire data bus. This technique is usually referred to as data bus division. In this case, the data and check bits of one ECC codeword are distributed to these subarrays, and each subarray is equipped with its own redundancies [5]. This scenario certainly complicates repair analysis to some extent. Nevertheless, the repair analysis algorithm exploration in this paper assumes that all data and check bits in one ECC codeword are located in one subarray, as illustrated in Fig. 1.

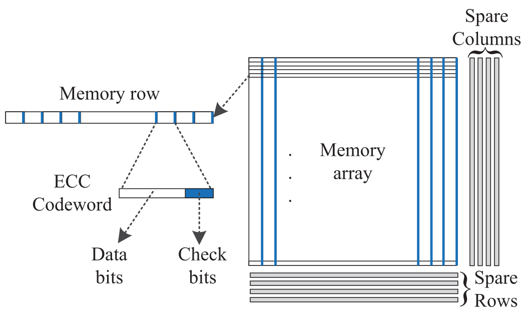


Fig. 1. Memory subarray protected by redundancy and in-memory ECC.

In-memory ECC has been proposed for yield enhancement very early and the synergistic effect of combining redundancy repair and in-memory ECC was first reported in 1990 [28]. However, the overhead of in-memory ECC on DRAM area and performance was unacceptably high in the past (e.g., 50 percent for DRAMs that should operate in a unit of single byte [25]), which inhibited the wide adoption of in-memory ECC. Recently, as the error rate of SCDs increases with the scaling down of DRAM cells, SCDs take a very large percentage of on-die defects. It is very inefficient to repair a large amount of SCDs by redundant rows and columns. On the contrary, ECC can efficiently repair the distributed SCDs, except for two or more SCDs located in one codeword. Moreover, the ECC encoding and decoding speed has been significantly improved because of the advances on

TABLE 1
Comparison of Conventional Repair Analysis Algorithms

Category	Algorithm	Repair rate	Analysis speed
Exhaustive	Fault-driven [11]	Optimal	Very slow
	B&B [12]	Optimal	Slow
	PAGEB [34]	Optimal	Faster than B&B
	VERA [31]	Optimal	Faster than PAGEB
	FGPM [32]	Optimal	Faster than VERA
Heuristic	Repair-most [10]	Low	Fast
	FAST [13]	Medium	Very fast

fabrication technologies, and the access granularity per DRAM device has been increased according to the continuous growth of memory bandwidth requirement. These conditions have amortized the performance and area overhead of in-memory ECC and made it possible to implement in-memory ECC in DRAM devices. Therefore, in-memory ECC becomes attractive to memory design community and is promising to overcome DRAM scaling challenges [16], [17], [18]. DRAM manufacturers have presented several DRAM products and design prototypes employing in-memory ECC [19], [21], [22], [23].

There are also some works that propose to overcome DRAM scaling challenges through fault tolerant architecture design [15], [29], [30]. These architectures are technically feasible and cost-efficient. However, they inevitably need to considerably modify current DRAM architecture design and operating flow. In-memory ECC is fully compatible and easy to implement compared with these designs. In-memory ECC can also be employed for reliability enhancement. Several studies have explored how to effectively use in-memory ECC to improve the reliability of memory systems [14], [29]. Nevertheless, yield improvement is the primary purpose to integrate in-memory ECC [2], [16], [19], [21], [22], [28]. A repair analysis algorithm is a fundamentally needed technique when in-memory ECC is used or partially used to repair defects for yield enhancement. It is critical to the yield and overall cost efficiency of memory equipped with both redundancy and in-memory ECC [24], [31], [32].

2.2 Traditional Repair Analysis Algorithms

This paper aims to explore efficient repair analysis algorithms for memory equipped with redundancy and in-memory ECC. We briefly summarize conventional memory repair analysis algorithms in Table 1 and review them as follows. Readers can refer to [33] for more extensive survey on memory repair analysis algorithms. The complexity of spare allocation problem for memory with two-dimensional redundancies (spare rows and columns) has been proofed to be NP-complete [12]. Thus, exhaustive search-based algorithms [11], [12], [31], [32], [34], which considers all possible cases, are proposed to achieve the optimal repair rate. Day *et al.* [11] propose a fault-driven algorithm that searches for a repair solution by conducting binary search tree (BST). Each BST node records the repair solution of previously detected defects. When a new defect is detected, two solution records are created from each original solution record, where the first record is expanded to the row address, and the second record is expanded to the column address. This algorithm essentially uses a breadth-first search (BFS) strategy, where its computational complexity and memory space requirement exponentially grow (in the worst case) with the problem scale (i.e., the number of defects). Thus, this algorithm is very expensive and time-consuming for high density memory with high BER of SCDs.

To reduce the computational complexity of exhaustive search algorithm, several preprocessing and filter algorithms are proposed to either terminate the analysis procedure as early as possible or skip as many faulty cells as possible. The must-repair algorithms [10], [11] are proposed to repair the defective lines (rows/columns) that must be repaired to find a solution. A defective row must be replaced with a spare row when it contains more defects than the available spare columns. Similarly, a defective column must be repaired with a spare column if it contains more defects than the available spare rows. The early-abort algorithms [11], [12], [34], [35] are proposed to identify unrepairable memory arrays before repair analysis to reduce the time wasted on unrepairable memory arrays. In addition, many defects belong to single faulty cell that shares no address with any other defects and can be repaired by only one spare element, regardless of the spare type. Accordingly, single-faulty-cell filter algorithm is proposed to reduce the number of defects analyzed in repair analysis by recording and filtering out the single faulty cells. There are also some exhaustive search algorithms that have been proposed to achieve optimal repair rate while alleviating the complexity induced by BST. Kuo and Fuchs [12] propose a branch-and-bound (B&B) algorithm, that takes the difference between the cost of embedding a spare row and spare column into consideration. Lin *et al.* [34] propose a PAGEB algorithm, that transforms the repair analysis problem into Boolean functions handled by a binary decision diagram. In this way, PAGEB algorithm achieves optimal repair rate with relatively fast analysis speed. Cho *et al.* [31] develop a VERA algorithm, which can achieve optimal repair rate and relative high analysis speed by using fault grouping and conducting binary search in each group. Lee *et al.* [32] propose an FGPM algorithm, that also uses fault grouping but all repair cases are searched through fault group pattern matching rather than constructing binary search tree.

Besides exhaustive search repair analysis algorithms, there are also several heuristic algorithms [10], [13] that can process the spare allocation problem with high analysis speed at the penalty of non-optimal repair rate. Repair-most (RM) [10] is a well-known heuristic algorithm. RM algorithm arranges the defective rows and columns in descending order according to the number of defects in them, repairs the row/column with the greatest number of defects, and updates the number of uncovered defects of each row and column. This process is repeated until no defects remains. The memory array is unrepairable by RM algorithm if defects remain after all redundancies are used. FAST algorithm [13] uses the concept of defect grouping to improve the repair rate of RM algorithm. The defects are grouped into single-defect group and multiple-defect groups depending on their relations with the defect locations. All defects in a group are repaired with spares of the same type. The repair solutions are obtained by checking all possible combinations, which compare the number of required spares with the number of available spares.

There are also many built-in repair analysis (BIRA) methods [36], [37], [38] designed for the yield improvement of embedded memories. Since BIRA algorithms are usually implemented by hardware with resource constraints, they mainly focus on the area overhead and repair analysis speed. Nevertheless, this paper focuses on the software-based repair analysis algorithms running on automatic test equipment.

3 Evaluation on Straightforward Algorithms

This section aims to demonstrate the necessity of efficient repair analysis algorithm with optimal repair rate for memory with redundancy and in-memory ECC. It seems that, in-memory ECC provides additional fault-tolerant capability beside redundancy repair, hence the repair analysis problem should become easier, the conventional repair analysis algorithms with a slight modification should be sufficiently capable of solving the problem. To clarify this point, we first evaluate two straightforward algorithms by slightly modifying conventional repair analysis algorithms. As a convenience, we give the following two definitions: the codeword that has two or more defects is defined as ECC-uncorrectable-codeword (EUCW), and the defect in EUCW is defined as ECC-uncorrectable-defect (EUCD).

The two straightforward algorithms based on conventional repair analysis algorithms are described as follows.

- *Repair without considering in-memory ECC (R_w/o_E):* The redundancies are allocated by using conventional repair analysis algorithm to repair defects without considering the fault-tolerant capability of in-memory ECC. After the redundancy repair, in-memory ECC is invoked to tolerate the remaining defects. The memory is repairable if there is no EUCW remaining after repair analysis. Otherwise, the memory is unrepairable.
- *Repair all EUCDs (R_all_EUCDs):* The algorithm first identifies all EUCDs, then employs conventional repair analysis algorithm to search for the repair solution that covers all the EUCDs or as many EUCDs as possible with the equipped redundancies. Finally, the remaining EUCDs are further checked within each ECC codeword. The memory is repairable if all EUCWs become correctable after repair analysis. Otherwise, the memory is unrepairable.

We note that the conventional repair analysis algorithm used in the above two algorithms can be either exhaustive search or RM. Nevertheless, when redundancy repair can not cover all the defects and in-memory ECC is invoked to correct the remained defects, RM has better performance than exhaustive search. This is because that RM is a greedy algorithm, and tends to cover more defects than exhaustive search algorithms, if the repair solution does not exist. Therefore, we employ RM in the above two algorithms during the evaluation. We also do not take BIRA algorithms into consideration, as BIRA algorithms are usually designed for embedded memory, and mainly aim to achieve low cost repair analysis at the penalty of certain repair rate loss [36], [39], [40]. While this paper aims to exploit the efficient repair analysis algorithm for high-density commodity DRAM, where repair rate is the primary design consideration [11], [12], [31], [32].

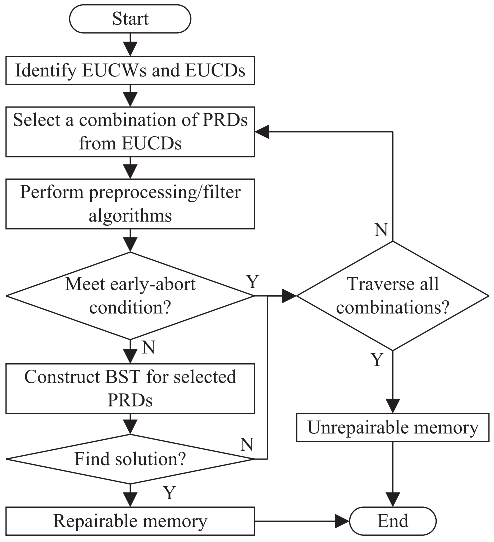

Start
Identify EUCWs and EUCDs
Select a combination of PRDs from EUCDs
Perform preprocessing/filter algorithms
Meet early-abort condition?
Y
N
Traverse all combinations?
N
Y
Construct BST for selected PRDs
Find solution?
N
Y
Unrepairable memory
Repairable memory
End

Fig. 2. Basic flow of straightforward exhaustive search repair analysis algorithm for memory with redundancy and in-memory ECC.

Then, we consider a straightforward algorithm to achieve optimal repair rate for memory with redundancy and in-memory ECC. Obviously, exhaustive search is a feasible solution. Fig. 2 illustrates the basic flow of the straightforward exhaustive search algorithm for memory with redundancy and in-memory ECC. First, all EUCWs and EUCDs are identified. If an EUCW contains N EUCDs, at least $N-1$ EUCDs should be repaired by a spare row or $N-1$ spare columns to make the remaining defect correctable by ECC. In this paper, we define the selected EUCDs in each EUCW as preemptive repair defects (PRDs), because they are preemptively repaired to make the memory array repairable. Thus, the algorithm exhaustively traverses all the possible combinations of PRDs and uses BST-based exhaustive search for each combination. If a repair solution is found before all the combinations have been traversed, the memory is repairable. Otherwise, the memory is unrepairable. To improve the analysis speed, we further employ the preprocessing/filter algorithms mentioned in Section 2.2 before conducting exhaustive search for each combination. The employed preprocessing/filter algorithms include must-repair, single-faulty-cell filter and early-abort algorithms. If bitmap of a PRDs combination meets the early-abort condition, the time-consuming BST traversal of the combination can be avoid. During the evaluation, we simulate the following two versions of this exhaustive-search-based straightforward algorithm.

- *Combination traversal with BST (Comb+BST):* This algorithm is the same as the operational flow presented in Fig. 2, where BST-based exhaustive search is employed to search the repair solution for each combination of selected PRDs.
- *Combination traversal with RM (Comb+RM):* This algorithm utilizes RM instead of BST-based exhaustive search to search for the repair solution in each combination of selected PRDs. This process can reduce the repair analysis time induced by BST traversal, but potentially has negative impact on repair rate.

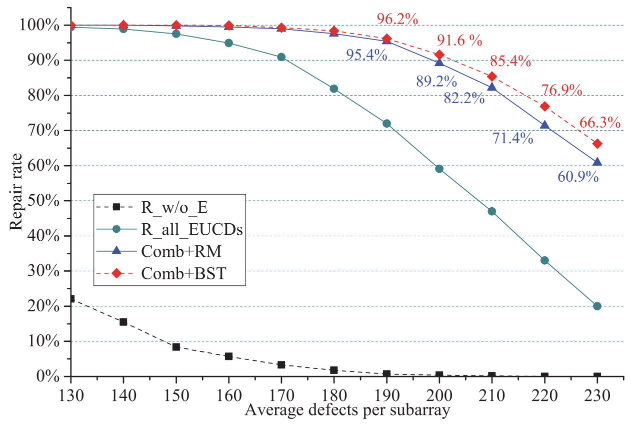
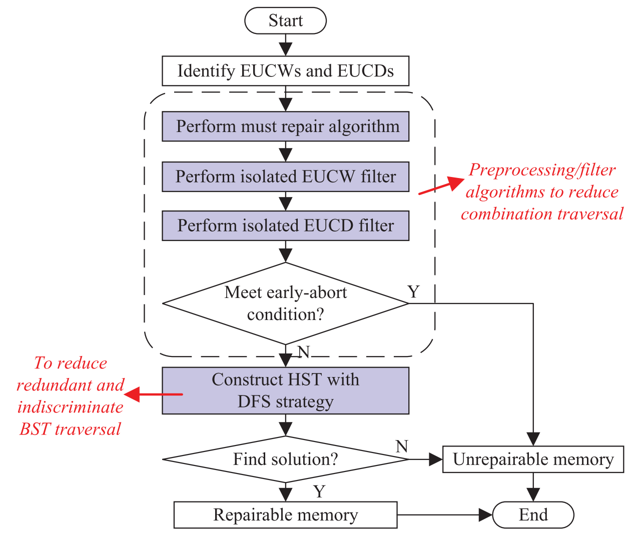


Fig. 3. Repair rate comparison among different straightforward repair analysis algorithms.



Start
Identify EUCWs and EUCDs
Perform must repair algorithm
Perform isolated EUCW filter
Perform isolated EUCD filter
Meet early-abort condition?
Y
N
Preprocessing/filter algorithms to reduce combination traversal
Construct HST with DFS strategy
To reduce redundant and indiscriminate BST traversal
Find solution?
N
Y
Unrepairable memory
Repairable memory
End

Fig. 4. Overall flow of the proposed optimal repair analysis algorithm.

We evaluate the repair rates of these four straightforward repair analysis algorithms using the experimental setup presented in Section 6.1. Fig. 3 shows the simulated repair rate results under different defect densities. The repair rate of R_w/o_E drastically decreases with the increase in defect density. This is because that R_w/o_E allocates redundant rows or columns without considering in-memory ECC. Many redundancies are allocated to repair ECC tolerable defects, whereas some EUCDs are uncovered. Although R_all_EUCDs can achieve much better repair rate than R_w/o_E, its repair rate considerably drops under high defect density compared with exhaustive search algorithms. The repair rate can be significantly improved by traversing all combinations of selected PRDs. Moreover, Comb+BST outperforms Comb+RM under high defect density, that the repair rate of Comb+BST is higher than that of Comb+RM by at least 5.5 percentage when defect density increases to 220 defects per subarray. The major drawback of Comb+BST and Comb+RM is their excessively high computational complexity making them infeasible for practical use. The experimental result in Section 6.2 shows that the average repair analysis time for one 512×544 memory subarray can be larger than $11s$ when the defect density increases to 180 defects per subarray. For an 8 Gb DRAM chip consists of 32K 512×544 subarrays, the average repair analysis time per chip is longer than 100 hours, which is unacceptable for practical memory repair analysis. The evaluation results on straightforward algorithms clearly demonstrate that it is worthwhile to explore an efficient repair analysis algorithm with optimal repair rate for memory with redundancy and in-memory ECC.

4 Proposed Optimal Repair Analysis Algorithm

4.1 Motivation and Overall Flow

Although the straightforward exhaustive search algorithm can achieve optimal repair rate for memory with redundancy and in-memory ECC, its computational complexity is extremely high, especially for high defect density scenarios. Hence, the analysis speed of this straightforward algorithm is excessively slow for practical use. To explore an efficient algorithm, we analyze that the high computational complexity of the exhaustive search algorithm is mainly incurred by the following three reasons.

- Excessive Combination traversal: The number of combinations of selected PRDs is the Cartesian product of the number of all possible selections of PRDs for each EUCW, which exponentially grows with the number of EUCWs in the worst case. Thus, traversing all combinations is extremely time-consuming for high-density memory with high defect density.
- Redundant BST traversal: For each combination, the exhaustive search algorithm has to repetitively construct the BST from a certain root node. This process is highly redundant because different combinations tend to have a large percentage of common PRDs. This redundant operation consumes considerable time when it is not considered.
- Inappropriate search strategy: Essentially, the conventional BST-based repair analysis algorithm indiscriminately traverses all the BST nodes using the BFS strategy. This process makes the computational complexity of traversing all nodes of BST exponentially grow with the number of defects in the worst case. The computational complexity can be significantly reduced with better search strategies.

Therefore, this paper exploits an efficient repair analysis algorithm with optimal repair rate by optimizing the exhaustive search algorithm according to the above three reasons. The overall flow of the proposed optimal repair analysis algorithm is shown in Fig. 4. The operational flow of the proposed algorithm is similar to that of exhaustive search. Nevertheless, several techniques are employed to reduce the computational complexity and speed up the repair analysis. In particular, isolated EUCW filter, isolated EUCD filter, and early-abort algorithm are designed to reduce the combination traversal, HST is proposed to reduce the redundant BST traversal, and DFS strategy is used to accelerate the solution search with prior knowledge. We explain each technique in detail in the following subsections.

4.2 Preprocessing/Filter Algorithms

4.2.1 Must-Repair Algorithm

Similar to the conventional must-repair algorithm, the criteria of the proposed must-repair process are based on the

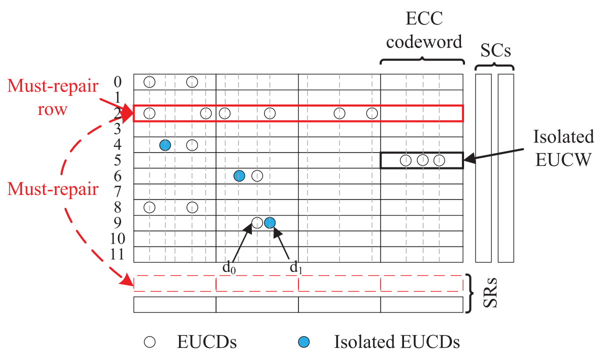


Fig. 5. Example of the proposed must-repair, isolated EUCW filter, and isolated EUCD filter algorithms.

number of available spare rows and columns in the subarray. Nevertheless, the proposed must-repair algorithm focuses on EUCDs and EUCWs instead of defects. To perform must-repair analysis, a row EUCW counter and a row EUCD counter are created for each defective row in the subarray. If the number of $(N_{EUCD} - N_{EUCW})$ in a row is larger than available spare columns, the defective row must be repaired by a spare row. Although there are also some must-repair columns, the proposed must-repair algorithm does not analyze the defective columns, since the repair to certain defective column is optional within each ECC codeword. This is the major difference between the conventional and the proposed must-repair algorithms.

As the example shown in Fig. 5, the defective memory subarray contains 9 EUCWs. In row “2”, which is labeled with “Must-repair row”, six EUCDs are distributed in three EUCWs. At least $N_{EUCD} - N_{EUCW} = 3$ spare columns are needed to repair these EUCWs, while there are only two available spare columns. Thus, these EUCWs must be repaired with one spare row.

4.2.2 Isolated EUCW Filter

An EUCW with all its EUCDs share no address with any other EUCD is defined as a isolated EUCW in this paper. Similar to the single faulty cell in conventional repair analysis, the repair solution to isolated EUCW is deterministic that one isolated EUCW can be independently repaired with a spare row or $N-1$ spare columns, where N represents the number of EUCDs in that isolated EUCW. As shown in Fig. 5, the EUCW in row “5”, which is labeled with “Isolated EUCW”, is an isolated EUCW because no other EUCW is found in row “5” and all EUCDs in it do not share column with other EUCDs. The repair solution for this EUCW is deterministic, either with one spare row, or with two spare columns, and does not affect the repair result of other EUCWs. Based on this observation, we propose an isolated EUCW filter that reduces the number of EUCWs to be analyzed in the combination traversal by recording and filtering out isolated EUCWs. Once the following repair analysis has found a potential solution for the non-isolated EUCWs, the remaining spare lines can be easily calculated. In particular, if the recorded isolated EUCWs can be repaired with the remaining spare rows and columns, a repair solution can be found. Otherwise, the repair analysis should continue to find other candidate repair solutions. The use of isolated EUCW filter can effectively reduce the number of combinations to be traversed during repair analysis.

4.2.3 Isolated EUCD Filter

An EUCD that locates in a non-isolated EUCW and does not share column address with any other EUCD is defined as an isolated EUCD in this paper, as shown in Fig. 5. We propose to filter out the isolated EUCDs to further reduce the number of combinations to be traversed during repair analysis. We have the following theorem.

Theorem 1. *To filter out isolated EUCDs in each EUCW from being selected as PRDs does not have any negative impact on the final repair rate.*

Proof. Let $d_1, d_2, \ldots, d_n$ denote n EUCDs in a EUCW, d_1 denote an isolated EUCD, and at least $n-1$ defects need to be repaired by redundancy in the EUCW. If the exhaustive search finds a repair solution when $n-1$ defects including d_1 are selected to be PRDs in the EUCW, it is not difficult to find that the repair solution also exists when d_1 is replaced by the remaining defect with column-sharings. On the contrary, if the exhaustive search finds a repair solution when selecting $n-1$ defects except for d_1 as PRDs in the EUCW, the memory subarray may become unrepairable when one of the defects is replaced by d_1. Thus, the theorem is proven. □

The theorem can be further demonstrated with the example shown in Fig. 5. Let us take the repair of the EUCW in row “9” as an example. There are two EUCDs, i.e., d_0 and d_1, in the EUCW, where d_1 is an isolated EUCD after row “2” has been repaired with spare row during must-repair analysis. Thus, the EUCW can be repaired either with one spare row or with one spare column. If a repair solution can be found when the EUCW is repaired with one spare row, the repair solution is constantly valid no matter which EUCD in the EUCW is selected as PRD. If there is one repair solution S_1 in which d_1 is selected to be PRD and is repaired with one spare column, another repair solution S_2 can certainly be found if d_0 is selected to be PRD. In the worst case, S_2 can be formed by keeping the repair of other EUCWs unchanged and allocating the spare column for d_1 in S_1 to repair d_0. In return, if there is no repair solution can be found if d_0 is selected as PRD, selecting d_1 as PRD cannot improve the condition. In one word,filtering out d_1 from being selected to be PRD does not impact the repair analysis result of the memory subarray.

In addition, although EUCDs with column-sharings have higher priority to be repaired than that of isolated EUCDs, it should not be concluded that EUCDs with more column-sharings also have higher priority than that with less column-sharings. This point can be easily proved through contradiction, as shown in Fig. 6. In this example, if we give higher repair priority to the PRDs with the most column-sharings, the memory subarray is unrepairable. Nevertheless, the memory subarray is actually repairable when we repair the PRDs with less column-sharings. With Theorem 1, we can safely employ the isolated EUCD filter before combination traversal to further reduce the computational complexity of the repair analysis for memory with both redundancy and in-memory ECC.

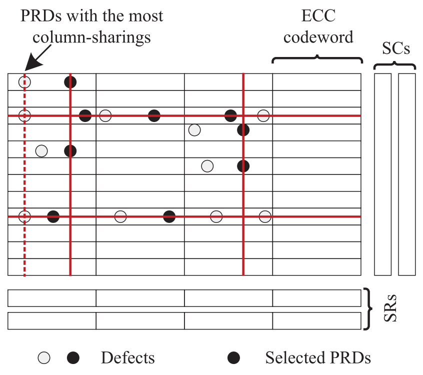


Fig. 6. Example to demonstrate that the combination of less column-sharings outperforms that of the most column-sharings.

4.2.4 *Early-Abort Algorithm*

The early-abort algorithm is employed to detect unrepairable memory subarray in advance during repair analysis. The proposed early-abort algorithm focuses on the must-repair columns untouched in the proposed must-repair algorithm. A column is called a must-repair column when the number of EUCDs in it is larger than the number of available spare rows after must-repair. If a column of EUCWs contains only one must-repair column, the EUCDs on this column may become ECC correctable defects after redundancy repair to other EUCDs. We develop an early-abort algorithm by utilizing the columns of EUCWs that contain more than one must-repair columns, which are called must-repair EUCW columns in brief. For a must-repair EUCW column that involves N must-repair columns, at least $N-1$ defective columns must be repaired with spare columns. We denote the number of must-repair EUCW columns in the subarray as T_w, the number of must-repair columns in these must-repair EUCW columns as T_c, and the number of available spare columns as SC. Then, at least $T_c - T_w$ spare column are needed to make the memory subarray repairable. Thus, the memory subarray is unrepairable when $T_c - T_w > SC$.

Fig. 7 illustrates an example of memory subarray that meets the condition of the proposed early-abort algorithm. After must-repair analysis, there are two must-repair EUCW columns, i.e., $T_w = 2$, and the number of must-repair columns involved in these must-repair EUCW columns is 5, i.e., $T_c = 5$. For the first must-repair EUCW column, column "a" or "b" must be repaired with one spare column, and for the second must-repair EUCW column, two of the three must-repair columns ("c", "d", and "e") must be repaired with two spare columns. Thus, the minimum number of required spare columns is three ($T_c - T_w = 3$), which is greater than the number of available spare columns. Hence, the memory subarray is unrepairable.

4.3 Hybrid Search Tree

To reduce redundant BST traversals, we propose a HST to complete the combination and BST traversals with one data structure. As shown in Fig. 8, different from traditional BST, the proposed HST has two kinds of nodes, i,e., combination nodes and repair nodes. For each selected EUCW, a group of combination nodes are created to traverse all the possible combinations of selected PRDs. Then

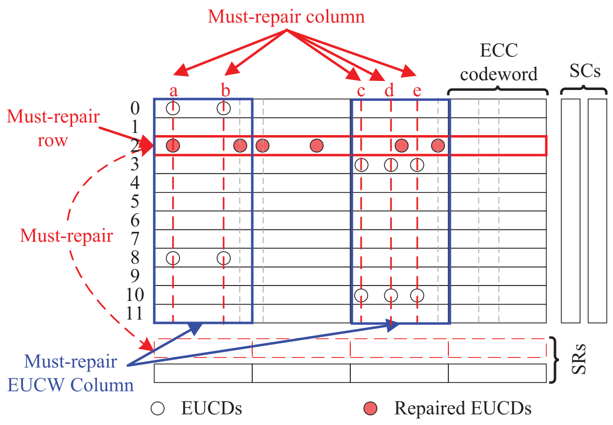


Fig. 7. Example of memory subarray that meets the condition of the proposed early-abort algorithm.

the repair nodes are created from these combination nodes. The number of combination nodes is N for an EUCW containing N EUCDs. Originally, the number of the corresponding repair nodes for each combination node is two, one for spare row and the other for spare columns. Nevertheless, as all the EUCDs in a EUCW can be repaired with a same spare row, the total number of repair nodes for the EUCW is $N+1$.

Fig. 8 also shows an example of the proposed HST. The memory subarray is 8×8 and is equipped with two spare rows and two spare columns. To simplify the example, we assume that each ECC codeword only consists of 4 bits while still can correct one error bit. The defective memory subarray contains 12 EUCDs that are randomly allocated in 5 EUCWs. Fig. 8a depicts the defective memory subarray after identifying EUCWs and EUCDs. We should note that, to simplify the description of the proposed HST structure, the proposed isolated EUCW filter and isolated EUCDs filter are not employed in the example. As shown in Fig. 8b, after EUCW in row "0" is selected to be repaired, the algorithm checks whether the selected EUCW has become ECC correctable under the spare allocation recorded in node "1". Because the selected EUCW has not been covered by spare allocation recorded in node "1", three combination nodes (i.e., nodes "2", "3", "4") are created from node "1", since the selected EUCW involves three EUCDs. For combination node "2" that consists of two selected PRDs located at (0, 0) and (0,1), nodes "5" and "6" are created to repair the selected PRDs with one spare row or two spare columns. Similarly, nodes "7" and "8" are created from nodes "3" and "4", respectively. Since, all EUCDs in the selected EUCW can be repaired with one spare row, node "5" is the common child node of nodes "2", "3" and "4". Figs. 8c, 8d, 8e, and 8f repeat this hybrid search for the remaining EUCWs. Finally, the HST finds five repair solutions, as shown in Fig. 8f.

The computational complexity of repair analysis can be significantly reduced by using the proposed HST instead of Comb+BST. Let T and W denote the total number of EUCDs and EUCWs in a subarray, respectively. The number of EUCDs in the ith EUCW is denoted as D_i, thus $T = \sum_{i=1}^{W} D_i$. If we do not take preprocessing and filter algorithms into consideration, Comb+BST has to traverse $\prod_{i=1}^{W} D_i$ combinations, and $(2^{(T+1)} - 1)$ repair nodes have to be traversed for each combination in the worst case. Thus, the total number

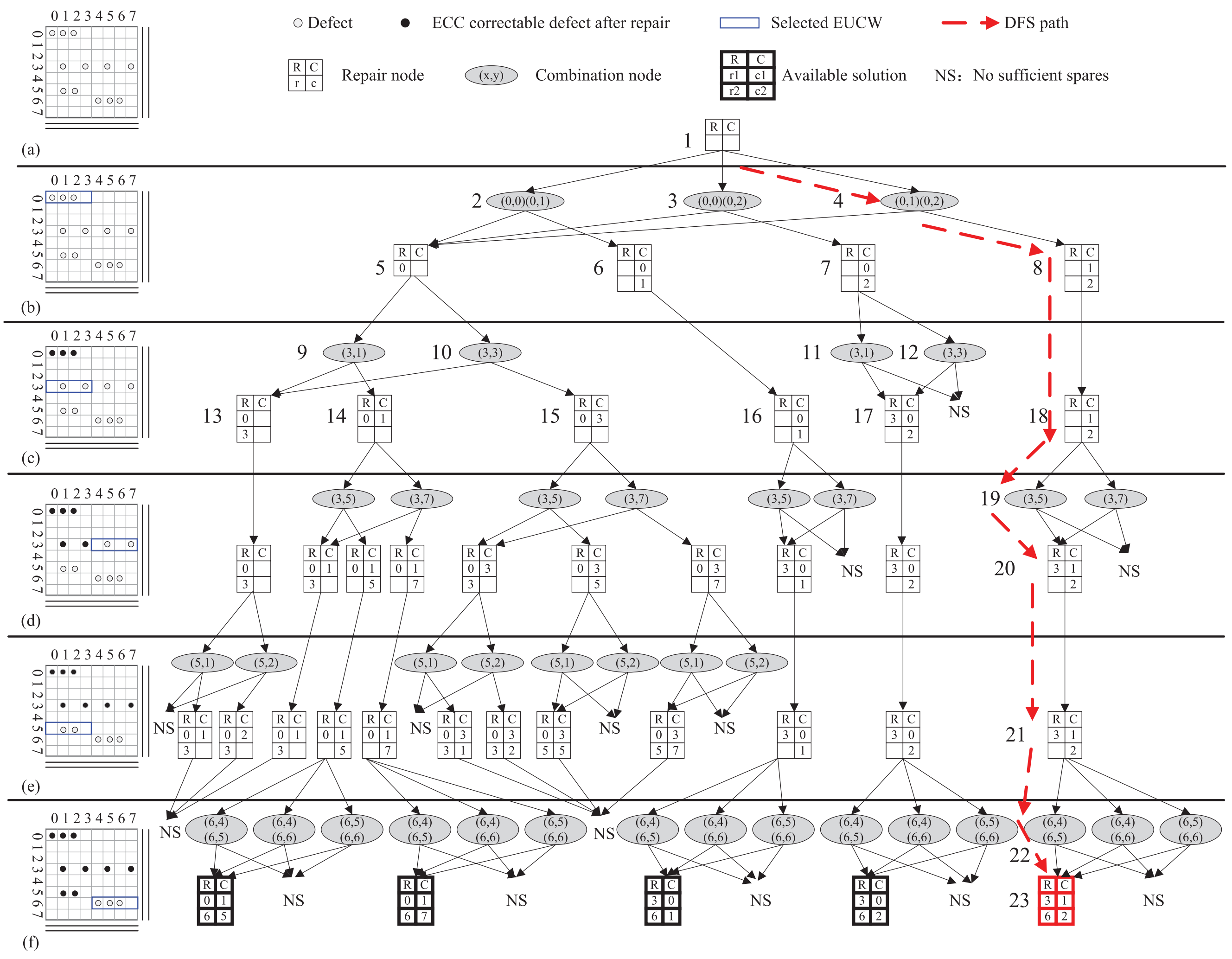


Fig. 8. Example of the proposed hybrid search tree with depth-first search strategy.

of nodes need to be traversed through Comb+BST can be approximately calculated using Equation (1)

$$N_{Comb+BST} = (2^{(T+1)} - 1) \cdot \prod_{i=1}^{W} D_i. \tag{1}$$

The number of nodes that need to be traversed using the proposed HST can be approximately calculated using Equation (2), where D_0 is set to 0

$$\begin{aligned} N_{HST} =& (2D_1 + 1) + (2D_2 + 1)(D_1 + 1) \\ &+ (2D_3 + 1)(D_2 + 1)(D_1 + 1) + \cdots \\ =& \sum_{k=1}^{W} [(2D_k + 1) \cdot \prod_{i=0}^{k-1} (D_i + 1)], \end{aligned} \tag{2}$$

N_{HST} is much smaller than $N_{Comb+BST}$, especially when the total numbers of EUCDs and EUCWs are large. This comparison can be quantitatively demonstrated using the simple example shown in Fig. 8, where $T = 12, W = 5, D_i = \{3, 2, 2, 2, 3\}$. Thus, in the worst case, $N_{comb+BST}$ and N_{HST} can reach up to 589,752 and 1,123, respectively. Therefore, the computational complexity in terms of search nodes can be reduced by more than 500× on the use of the proposed HST instead of Comb+BST. In the practical traversal, since many branches terminate early due to the lack of spares and the corresponding nodes are eliminated, the numbers of nodes for both algorithms are much smaller than the calculated results. In the matter of fact, there are only 76 nodes (38 combination nodes and 38 repair nodes) for the proposed HST as shown in Fig. 8. For Comb+BST algorithm, there would be 72 combinations to be traversed, and for each combination, more than 100 BST nodes needed to be traversed for 7 selected PRDs. Thus, there would be up to 7,200 BST nodes that have to be traversed. The practical computational complexity reduction is still around 100×.

4.4 Depth-First Search Strategy

Conventional BST-based exhaustive search algorithms use the BFS strategy, which is actually inappropriate for repair analysis. In general, BFS strategy is preferred for BST when the repair solution can be found close to the root node where the search begins. This indicates the repair capability of redundancies largely exceeds the number of defects. However, redundancies are adequately equipped according to defect density in practical applications. Therefore, most of repair solutions should be found far from root node when redundancies are nearly used up. In this case, the BFS strategy is very inefficient in term of repair analysis time. More importantly, we can determine the branch or node

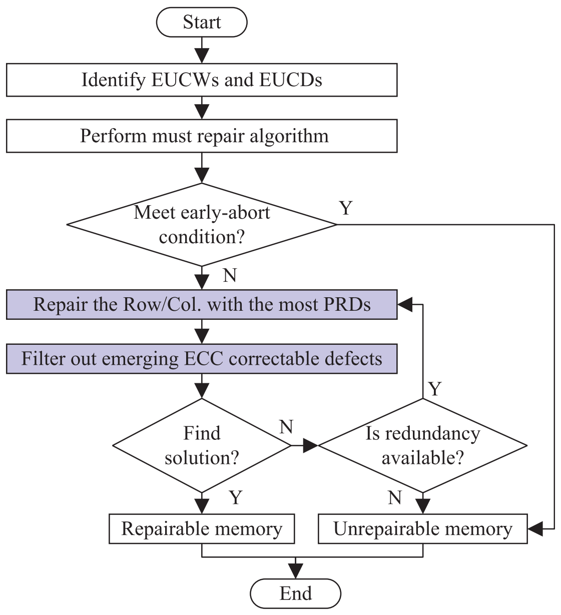

Start
Identify EUCWs and EUCDs
Perform must repair algorithm
Meet early-abort condition?
Y
N
Repair the Row/Col. with the most PRDs
Filter out emerging ECC correctable defects
Find solution?
N
Is redundancy available?
Y
Y
N
Repairable memory
Unrepairable memory
End

Fig. 9. Flowchart of the proposed heuristic repair analysis algorithm.

that is likely to contain the repair solution in advance by leveraging heuristic search algorithms (e.g., RM). Although heuristic search algorithms are not optimal, they can reach to repair solution in most cases. Therefore, we propose to employ the DFS strategy for HST traversal to improve the repair analysis speed.

As shown in Fig. 8, the dashed arrows illustrate the branches and nodes selected by the DFS strategy when traversing HST. In particular, we use a greedy algorithm for the DFS strategy, where the EUCW with the most row-sharings or column-sharings is selected to create combination and repair nodes, the branch with the most sharings is selected for each combination node, and the branch covering the most PRDs is selected for each repair node. We should note that, the number of PRDs on a column equals the number of EUCDs on the column, whereas the number of PRDs on a row equals the number of $(N_{EUCD} - N_{EUCW})$. Therefore, in the example shown in Fig. 8, combination node "4" is traversed with the highest priority because it has more column-sharings than that of combination nodes "2" and "3". Repair node "8" is traversed with higher priority than node "5" because node "8" covers five PRDs, whereas node "5" only covers two RPDs. Using this greedy DFS strategy, a repair solution can be found through the first DFS path ("1" →" 4" →" 8" →" 18" →" 19" →" 20" →" 21" →" 22" →" 23") . If the DFS path can not find a repair solution before reaching to the ending node, it can track back to other nodes or branches to find repair solution until all the HST nodes are traversed. We can see that, in the best case, the DFS strategy only need to traverse H nodes to reach the available solution node, while the BFS strategy has to traverse at least 2^H nodes, where H is the depth of the available solution node. In the worst case, both BFS and DFS strategies need to traverse all nodes in the tree, thus have the same computational complexity. By employing DFS strategy in HST traversal, we can not only significantly improve the repair analysis speed, but also reduce the memory footprint.

5 Proposed Heuristic Repair Analysis Algorithm

During the experiment of DFS strategy, we find that the majority of repair solutions can be found using the greedy algorithm, which prefers to choose the combination with the most column-sharings and repair the defective rows or columns with the most PRDs. This further motivates us to develop a heuristic repair analysis algorithm for memory with both redundancy and in-memory ECC. Memory fabrication cost depends on both repair rate and repair analysis speed, and repair analysis algorithm design usually has to balance between them. Although heuristic repair analysis algorithms usually can not provide optimal repair rate, they can easily achieve high repair analysis speed with low implementation cost and are useful in many cost-sensitive scenarios.

The proposed heuristic repair analysis algorithm is essentially very simple, and its flowchart is illustrated in Fig. 9. After must-repair and early-abort algorithms, the proposed heuristic algorithm recursively repairs the row or column with the most PRDs and removes the emerging ECC correctable defects after repair, until a repair solution is found or all the redundancies are used up. If there are remaining EUCDs when the redundancies are used up, the memory subarray is unrepairable with the heuristic repair analysis algorithm. The proposed heuristic algorithm can further speed up repair analysis, and its computational complexity is $\mathcal{O}(N)$ in the worst case, where N is the number of EUCDs. We will evaluate the repair rate and analysis speed of the proposed heuristic repair analysis algorithm in the next section.

6 Experimental Evaluation

6.1 Experimental Setup

Experiments are conducted based on the same memory subarray architecture illustrated in Fig. 1. The size of memory subarray is set as 512×544, indicating that it has 512 rows and 544 columns. Hence, the total data bits in the memory subarray are around 272 Kb. We assume that the memory subarray is equipped with (136, 128) shortened Hamming code as in-memory ECC. Each 136-bit codeword consists of 128 data bits and 8 check bits, and can only correct one error bit. Each memory row contains four ECC codewords, and the memory subarray contains a total of 2,048 ECC codewords. In addition, we assume that the memory subarray is equipped with 6 spare rows and 6 spare columns for memory repair. (i.e., SR = 6, SC = 6).

To evaluate the repair rate and computational complexity of different repair analysis algorithms, we build a simulator with the following characteristics. This simulator can support us to inject defects into random locations across the memory subarray, and can report the repair rate and the average repair analysis time per subarray of different repair analysis algorithms. The negative binomial distribution defect model [41] is used to generate the number of defects to be injected into each subarray under a given defect density. The percentage of SCDs, row cluster defects and column cluster defects are set to 95, 2.5 and 2.5 percent, respectively. Under each defect density scenario, 1,000 instances are simulated for each repair analysis algorithm.

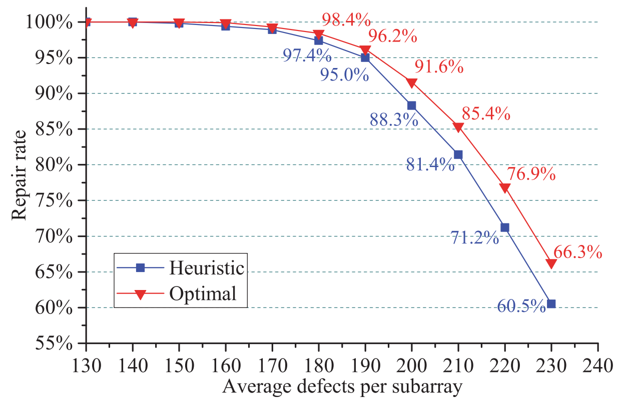


Fig. 10. Repair rate comparison between the proposed optimal and heuristic algorithms.

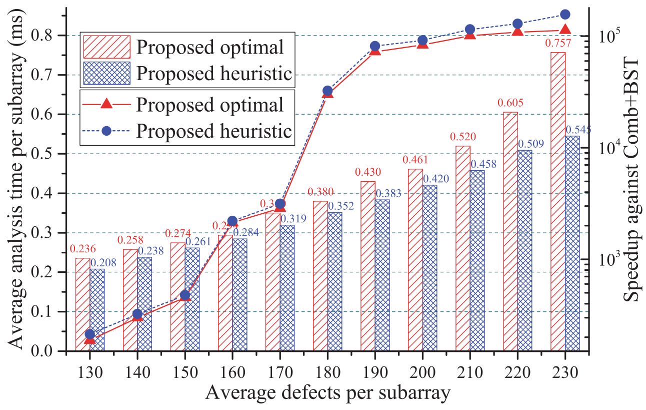


Fig. 12. Analysis time comparison between the proposed optimal and heuristic algorithms.

To quantitatively compare the computational complexity and analysis speed of different analysis algorithms, all algorithms are implemented and run as software on the same computer platform. The computer platform has 16 GB main memory and 4 Intel(R) Xeon(R) CPU running at 2.4 GHz frequency. The operating system is a Linux system with kernel version 4.15.0. All repair analysis algorithm codes are written in C++ programming language and compiled on GNU g++ 7.3.0 with the configuration of the optimization flag -O3. The repair analysis time of each algorithm is calculated by taking the average repair analysis time of 1,000 instances.

6.2 Experimental Results

Fig. 10 illustrates the simulated repair rate comparison between the proposed optimal and heuristic repair analysis algorithms. We should note that, the proposed optimal repair analysis algorithm has the same repair rate as that of exhaustive search algorithm (Comb+BST). The repair rate of the proposed heuristic repair analysis algorithm is slightly lower than that of Comb+RM algorithm. This is mainly because the proposed heuristic repair analysis algorithm essentially employs a greedy strategy to replace both combination and BST traversals. As shown in Fig. 10, the proposed optimal algorithm exhibits higher repair rates than that of the proposed heuristic algorithm with the increase in defect density. When the defect density increases to 230 defects per subarray, the repair rate of the proposed optimal algorithm outperforms that of the proposed heuristic algorithm by up to 5.8 percent. The repair rate comparison results clearly demonstrate the necessity of the optimal repair analysis algorithm. The results also show that the proposed heuristic algorithm can provide acceptable repair rate in spite of its simplicity.

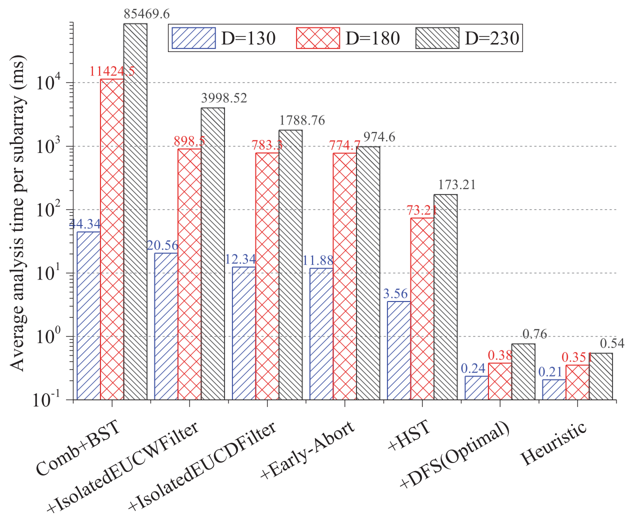
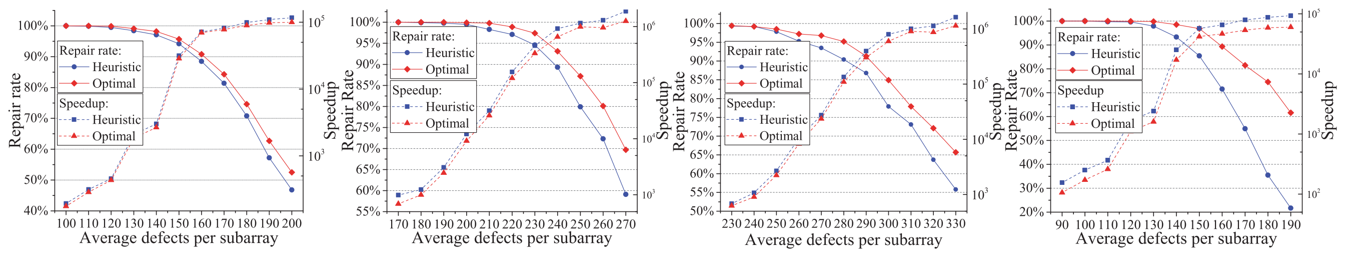


Fig. 11. Step-by-step computational efficiency analysis of the proposed algorithms at defect densities of 130, 180, and 230.

Fig. 11 illustrates the step-by-step computational efficiency analysis of the proposed algorithms at defect densities of 130, 180, and 230. We can see that, each proposed technique effectively improves the computational efficiency and hence significantly reduces the repair analysis time. More importantly, the proposed optimal repair analysis algorithm can achieve low repair analysis time that is comparable to the proposed heuristic algorithm by using these efficient techniques. Fig. 12 illustrates the analysis time comparison between the proposed optimal and heuristic algorithms by varying defect density from 130 to 230. The speedups of the two algorithms against Comb+BST are also illustrated. The repair analysis time of the proposed optimal algorithm remains comparable to that of the proposed heuristic algorithm. When the defect density increases to 230 defects per subarray, the proposed optimal algorithm only takes 0.757 ms on average for the repair analysis of each subarray, while the proposed heuristic algorithm is approximately 28 percent faster, at the expense of 5.8 percent repair rate drop. Moreover, the two proposed algorithms can significantly speed up the repair analysis against Comb+BST that the speedups drastically increase with the increase in defect density, and the speedups reach up to more than $10^5\times$ when the defect density is larger than 200. This further demonstrates the computational efficiency of the two proposed algorithms.

The defect density seems high in the experimental setup. However, redundancy repair is only necessary to EUCWs because each ECC codeword can correct one bit error. There are 2,048 ECC codewords in each memory subarray, and we assume that all defects are randomly distributed throughout each memory subarray with uniform probability under each defect density. Thus, the number of EUCWs in each subarray is excessively small even when the number of defects



(a) ECC (136,128), SR = 4, SC = 4 (b) ECC (136,128), SR = 8, SC = 8 (c) ECC (72, 64), SR = 6, SC = 6 (d) ECC (272, 256), SR = 6, SC = 6

Fig. 13. Repair rate and speedup against Comb+BST on different configurations of ECC codeword and redundancy.

per subarray is more than 200. In practice, the BER largely varies in terms of inter- and intra-subarrays. Therefore, the number of EUCWs in each subarray can be much larger even under lower average BER in practical scenarios.

Moreover, we should note that the choice of ECC codeword length and the number of redundancies in each subarray is largely determined on the basis of defect density. The proposed repair analysis algorithms are not limited to certain configuration of ECC codeword and redundancy, and can flexibly support repair analysis on different configurations. To demonstrate this, we conduct experiments to evaluate the repair rate and analysis speed of the proposed repair analysis algorithms on four additional cases with different ECC codeword length and redundancy configurations.

- Case 1: ECC (136,128) with SC = 4 and SR = 4
- Case 2: ECC (136,128) with SC = 8 and SR = 8
- Case 3: ECC (72,64) with SC = 6 and SR = 6
- Case 4: ECC (272,256) with SC = 6 and SR= 6

Fig. 13 illustrates the experimental results of repair rate and speedup against Comb+BST on the four aforementioned cases. The proposed repair analysis algorithms work well on all the four cases in terms of repair rate and analysis speed. The results demonstrate that the proposed repair analysis algorithms are insensitive to ECC codeword and redundancy configurations.

7 Conclusion

This paper aims to explore the efficient repair analysis algorithm for memory equipped with redundancy and in-memory ECC. In particular, we propose two repair analysis algorithms, namely, an optimal algorithm and a heuristic algorithm. The optimal repair analysis algorithm can achieve the optimal repair rate while reducing repair analysis time by up to $10^5\times$ compared with the exhaustive search algorithm. The heuristic algorithm is 28 percent faster than the optimal algorithm, at the expense of 5.8 percent repair rate drop. The experimental results demonstrate the effectiveness and efficiency of the proposed repair analysis algorithms. Nevertheless, the proposed algorithms can be only applied to the memory architecture where the entire ECC codeword, including all its data and check bits, is located in one subarray. Our future work will focus on exploring the efficient repair analysis algorithm for memory designed with data bus division technique and protected by redundancy and in-memory ECC.

Appendix A
Analysis on the Proposed Preprocessing/Filter Algorithms

The proposed preprocessing/filter algorithms reduce the computational complexity by reducing the number of EUCWs and EUCDs to be analyzed during the tree-based exhaustive search. This appendix evaluates the effectiveness of these algorithms approximately using probability analysis. Table A shows the acronyms and definitions used in the analysis.

We assume that the defects are randomly distributed across the memory subarray with uniform probability. Then, the number of defects (d) in each ECC word follows the binomial distribution, i.e., $d \sim B(L, \rho)$. Thus, the probability that an ECC word is an EUCW (P_{EUCW}) can be calculated with Equation (3). The probability that a cell is an EUCD (P_{EUCD}) can be calculated with Equation (4)

$$P_{EUCW} = \sum_{d=2}^{L} \left\{ \binom{L}{d} \rho^d (1-\rho)^{(L-d)} \right\} \tag{3}$$

$$P_{EUCD} = \rho \cdot [1 - (1-\rho)^{(L-1)}]. \tag{4}$$

Similarly, the number of EUCWs in each row ($N_{EUCW/r}$) follows the binomial distribution, i.e., $N_{EUCW/r} \sim B(N_{w/r}, P_{EUCW})$.

A.1 Must Repair

A row is a must-repair row when $N_{EUCD/r} - N_{EUCW/r} > SC$. Thus, its probability can be calculated as

$$P(r \in S_{MR}) = \sum_{i=1}^{N_{w/r}} \{ P(N_{EUCW/r} = i) \cdot P(r \in S_{MR} | N_{EUCW/r} = i) \}. \tag{5}$$

TABLE A
Acronyms and Definitions

N_r	the number of rows in each subarray
$N_{w/r}$	the number of ECC words in each row
L	the ECC word length
ρ	BER of defects
SC:	the number of spare columns in each subarray
SR:	the number of spare rows in each subarray
S_{EUCW}	set of EUCWs in subarray
S_{IEUCW}	set of isolated EUCWs in subarray
S_{EUCD}	set of EUCDs in subarray
S_{MR}	set of must-repair rows in subarray

Where, $P(N_{EUCW/r} = i)$ is the probability that the number of EUCWs in a row is i and can be calculated as

$$P(N_{EUCW/r} = i) = \binom{N_{w/r}}{i} P^{i}_{EUCW} \cdot (1 - P_{EUCW})^{(N_{w/r}-i)}. \tag{6}$$

$P(r \in S_{MR} | N_{EUCW/r} = i)$ is the probability that a row is a must-repair row under the condition that the row contains i EUCWs. The formulations for $P(r \in S_{MR} | N_{EUCW/r} = i)$ vary with $N_{EUCW/r}$. When $N_{EUCW/r} = 1$, $P(r \in S_{MR} | N_{EUCW/r} = 1)$ can be calculated as

$$P(r \in S_{MR} | N_{eucw/r} = 1) = 1 - \sum_{d=0}^{SC+1} \binom{L}{d} \rho^{d} (1-\rho)^{(L-d)}. \tag{7}$$

When $i \geq 2$, $P(r \in S_{MR} | N_{EUCW/r} = i)$ can be calculated in the similar way.

A.2 Isolated EUCW Filter

An isolated EUCW is defined as an EUCW with all its EUCDs share no address with any other EUCDs. According to the Bayes' Theorem, the probability that an EUCW is an isolated EUCW can be calculated as a conditional probability

$$P(w \in S_{IEUCW} | w \in S_{EUCW}) = \frac{P(w \in S_{IEUCW})}{P(w \in S_{EUCW})}. \tag{8}$$

Where $P(w \in S_{IEUCW})$ and $P(w \in S_{EUCW})$ are the probability that an ECC word is an isolated EUCW and an EUCW, respectively. $P(w \in S_{IEUCW})$ can further be calculated with Equation (9)

$$P(w \in S_{IEUCW}) = (1 - P_{EUCW})^{N_{w/r}-1} \cdot \sum_{d=2}^{L} \{ \binom{L}{d} \rho^{d} \cdot (1-\rho)^{(L-d)} \cdot (1 - P_{EUCD})^{d(N_r - 1)} \}. \tag{9}$$

A.3 Isolated EUCD Filter

An isolated EUCD is defined as an EUCD that locate in the non-isolated EUCW and does not share column address with any other EUCD. Let $S_{D\_\overline{IEUCW}}$ and S_{IEUCD} denote the set of EUCDs located in non-isolated EUCW and the set of isolated EUCDs in a subarray. According to the Bayes' Theorem, the probability that an EUCD located in non-isolated EUCW is an isolated EUCD can be calculated with Equation (10)

$$P(x \in S_{IEUCD} | x \in S_{D\_\overline{IEUCW}}) = \frac{P(x \in S_{IEUCD})}{P(x \in S_{D\_\overline{IEUCW}})}. \tag{10}$$

$P(x \in S_{IEUCD})$ and $P(x \in S_{D\_\overline{IEUCW}})$ can be calculated with Equations (11) and (12), respectively

$$P(x \in S_{IEUCD}) = \sum_{d=2}^{L} \{ \binom{L}{d} \rho^{d} (1-\rho)^{L-d} \cdot (1 - P_{EUCD})^{(N_r - 1)} \cdot \frac{d}{L} \cdot [1 - (1 - P_{EUCW})^{(N_{w/r}-1)} \cdot (1 - P_{EUCD})^{(d-1)\cdot(N_r-1)}] \} \tag{11}$$

$$P(x \in S_{D\_\overline{IEUCW}}) = \sum_{d=2}^{L} \{ \binom{L}{d} \rho^{d} (1-\rho)^{L-d} \cdot \frac{d}{L} \cdot [1 - (1 - P_{EUCW})^{(N_{w/r}-1)} \cdot (1 - P_{EUCD})^{d\cdot(N_r-1)}] \}. \tag{12}$$

A.4 Early-Abort algorithm

The proposed early-abort analysis is based on the must-repair columns and must-repair EUCW columns. Thus, Equations (13) and (14) calculate the probabilities that a column is a must-repair column (P_{MRC}) and a column of ECC words is a must-repair EUCW column (P_{MRWC}), respectively

$$P_{MRC} = \sum_{d=SR+1}^{N_r} \binom{N_r}{d} \cdot P^{d}_{EUCD} \cdot (1 - P_{EUCD})^{(N_r - d)} \tag{13}$$

$$P_{MRWC} = \sum_{i=2}^{L} \binom{L}{i} \cdot P^{i}_{MRC} \cdot (1 - P_{MRC})^{(L-i)}. \tag{14}$$

Let S_{EA} denote the set of all the subarrays meet early-abort condition. According to the Bayes' Theorem, the probability that a subarray sa meets early-abort condition can be calculated with Equation (15)

$$P(sa \in S_{EA}) = \sum_{k=1}^{N_{w/r}} \{ P(sa \in S_{EA} | N_{MRWC} = k) \cdot P(N_{MRWC} = k) \}. \tag{15}$$

Where $P(N_{MRWC} = k)$ is the probability that a subarray contains k must-repair EUCW columns, which can be calculated with Equation (16)

$$P(N_{MRWC} = k) = \binom{N_{w/r}}{k} \cdot P^{k}_{MRWC} \cdot (1 - P_{MRWC})^{(N_{w/r}-k)}, \tag{16}$$

$P(sa \in S_{EA} | N_{MRWC} = k)$ is the probability that a subarray meets the proposed early-abort condition when it contains k must-repair EUCW columns. The formulations for $P(sa \in S_{EA} | N_{MRWC} = k)$ vary with k. When $k = 1$

$$P(sa \in S_{EA} | N_{MRWC} = 1) = 1 - \sum_{j=0}^{SC+1} \binom{L}{j} \cdot P^{j}_{MRC} \cdot (1 - P_{MRC})^{(L-j)}. \tag{17}$$

When $k \geq 2$, $P(sa \in S_{EA} | N_{MRWC} = k)$ can be calculated in the similar way.

A.5 Quantitative Analysis

Using the above equations, we evaluate the potential effectiveness of the proposed preprocessing/filter algorithms with the memory subarray configuration in Section 6.1. When the number of defects in each subarray is 130, the probability that a row is a must-repair row is around 7.8×10^{-9} , 97 percent of EUCWs and 58.36 percent EUCDs in non-isolated EUCWs could be filtered out from the tree-based exhaustive search, the probability that a subarray meets the proposed early-abort condition is 7.3×10^{-8}. When the number of defects in each subarray is 230, the probability that a row is a must-repair row is around 9.2×10^{-7}, 90 percent of EUCWs and 57.83 percent EUCDs in non-isolated EUCW could be filtered out from the tree-based exhaustive search, the probability that a subarray meets the proposed early-abort condition is 3.17×10^{-6}. We note that, although the probability that a memory subarray meets early-abort condition is low, it can cancel those cases incurring very heavy tree-based exhaustive search and its effectiveness is obvious. The results are roughly consistent with the simulation results in Section 6.2.

ACKNOWLEDGMENTS

This research was supported by National Natural Science Foundation of China under Grant 61722406 and Grant 61751401.

REFERENCES

[1] S. Hong, "Memory technology trend and future challenges," in *Proc. Int. Electron Devices Meet.*, 2010, pp. 12.4.1–12.4.4.

[2] S. Cha *et al.*, "Defect analysis and cost-effective resilience architecture for future DRAM devices," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2017, pp. 61–72.

[3] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design," in *Proc. Int. Symp. Archit. Support Program. Lang. Operating Syst.*, 2012, pp. 111–122.

[4] K. K. Hung, P. K. Ko, C. Hu, and Y. C. Cheng, "Random telegraph noise of deep-submicrometer MOSFETs," *IEEE Electron Device Lett.*, vol. 11, no. 2, pp. 90–92, Feb. 1990.

[5] M. Horiguchi and K. Itoh, "Redundancy," in *Nanoscale Memory Repair*. Berlin, Germany: Springer, 2011, pp. 19–67.

[6] C. H. Stapper and R. J. Rosner, "Integrated circuit yield management and yield analysis: Development and implementation," *IEEE Trans. Semicond. Manuf.*, vol. 8, no. 2, pp. 95–102, May 1995.

[7] H. Liu *et al.*, "A built-off self-repair scheme for channel-based 3D memories," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1293–1301, Aug. 2017.

[8] M. D. Smith and P. Mazumder, "Generation of minimal vertex covers for row/column allocation in self-repairable arrays," *IEEE Trans. Comput.*, vol. 45, no. 1, pp. 109–115, Jan. 1996.

[9] M. Chang, W. K. Fuchs, and J. H. Patel, "Diagnosis and repair of memory with coupling faults," *IEEE Trans. Comput.*, vol. 38, no. 4, pp. 493–500, Apr. 1989.

[10] M. Tarr, D. Boudreau, and R. Murphy, "Defect analysis system speeds test and repair of redundant memories," *Electronics*, vol. 57, no. 1, pp. 175–179, Jan. 1984.

[11] J. R. Day, "A fault-driven, comprehensive redundancy algorithm," *IEEE Des. Test Comput.*, vol. 2, no. 3, pp. 35–44, Jun. 1985.

[12] S. Kuo and W. K. Fuchs, "Efficient spare allocation for reconfigurable arrays," *IEEE Des. Test Comput.*, vol. 4, no. 1, pp. 24–31, Feb. 1987.

[13] H. Cho, W. Kang, and S. Kang, "A fast redundancy analysis algorithm in ATE for repairing faulty memories," *ETRI J.*, vol. 34, no. 3, pp. 478–481, Jun. 2012.

[14] P. J. Nair, V. Sridharan, and M. K. Qureshi, "XED: Exposing on-die error detection information for strong memory reliability," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit.*, 2016, pp. 341–353.

[15] Y. H. Son, S. Lee, O. Seongil, S. Kwon, N. S. Kim, and J. H. Ahn, "CiDRA: A cache-inspired DRAM resilience architecture," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 502–513.

[16] U. Kang *et al.*, "Co-architecting controllers and DRAM to enhance DRAM process scaling," in *Proc. Memory Forum*, Jun. 2014.

[17] "Low Power Double Date Rate 4 (LPDDR4)," JEDEC, vol. JESD209-4c, Jan. 2020. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd209-4b

[18] S. Kwon, Y. H. Son, and J. H. Ahn, "Understanding DDR4 in pursuit of in-DRAM ECC," in *Proc. Int. SoC Des. Conf.*, 2014, pp. 276–277.

[19] T.-Y. Oh *et al.*, "A 3.2 Gbps/pin 8 Gbit 1.0 V LPDDR4 SDRAM with integrated ECC engine for sub-1 V DRAM core operation," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 178–190, Jan. 2015.

[20] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, "Understanding and modeling on-die error correction in modern DRAM: An experimental study using real devices," in *Proc. Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2019, pp. 13–25.

[21] H. Kwon *et al.*, "An extremely low-standby-power 3.733Gb/s/pin 2Gb LPDDR4 SDRAM for wearable devices," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2017, pp. 394–395.

[22] K. C. Chun *et al.*, "A 16 Gb LPDDR4X SDRAM with an NBTI-tolerant circuit solution, an SWD PMOS GIDL reduction technique, an adaptive gear-down scheme and a metastable-free DQS aligner in a 10nm class DRAM process," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 206–208.

[23] N. Kwak *et al.*, "A 4.8 Gb/s/pin 2Gb LPDDR4 SDRAM with sub-100μA self-refresh current for IoT applications," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2017, pp. 392–393.

[24] H. Sun, J. Zhao, F. Wang, N. Zheng, and T. Zhang, "Cost-efficient built-in repair analysis for embedded memories with on-chip ECC," in *Proc. Int. Symp. Access Spaces*, 2011, pp. 95–100.

[25] S.-H. Kim *et al.*, "A low power and highly reliable 400 Mbps mobile DDR SDRAM with on-chip distributed ECC," in *Proc. IEEE Asian Solid-State Circuits Conf.*, 2007, pp. 34–37.

[26] C.-L. Su, Y.-T. Yeh, and C.-W. Wu, "An integrated ECC and redundancy repair scheme for memory reliability enhancement," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2005, pp. 81–89.

[27] K. Itoh, "High-performance subsystem memories," in *VLSI Memory Chip Des.*, vol. 5. Berlin, Germany: Springer, 2001, pp. 339–387.

[28] H. L. Kalter *et al.*, "A 50-ns 16-Mb DRAM with a 10-ns data rate and on-chip ECC," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1118–1128, Oct. 1990.

[29] S.-L. Gong, J. Kim, S. Lym, M. B. Sullivan, H. David, and M. Erez, "DUO: Exposing on-chip redundancy to rank-level ECC for high reliability," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 683–695.

[30] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates," in *Proc. ACM SIGARCH Comput. Archit. News*, 2013, pp. 72–83.

[31] H. Cho, W. Kang, and S. Kang, "A very efficient redundancy analysis method using fault grouping," *ETRI J.*, vol. 35, no. 3, pp. 439–447, Jun. 2013.

[32] H. Lee, K. Cho, D. Kim, and S. Kang, "Fault group pattern matching with efficient early termination for high-speed redundancy analysis," *IEEE Trans. Comput.-Aided De. Integr. Circuits Syst.*, vol. 37, no. 7, pp. 1473–1482, Jul. 2018.

[33] K. Cho, W. Kang, H. Cho, C. Lee, and S. Kang, "A survey of repair analysis algorithms for memories," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 1–41, Dec. 2016.

[34] H.-Y. Lin, F.-M. Yeh, and S.-Y. Kuo, "An efficient algorithm for spare allocation problems," *IEEE Trans. Rel.*, vol. 55, no. 2, pp. 369–378, Jun. 2006.

[35] R. W. Haddad, A. T. Dahbura, and A. B. Sharma, "Increased throughput for the testing and repair of RAMs with redundancy," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 154–166, Feb. 1991.

[36] C.-T. Huang, C.-F. Wu, J.-F. Li, and C.-W. Wu, "Built-in redundancy analysis for memory yield improvement," *IEEE Trans. Rel.*, vol. 52, no. 4, pp. 386–399, Dec. 2003.

[37] T. Kawagoe, J. Ohtani, M. Niiro, T. Ooishi, M. Hamada, and H. Hidaka, "A built-in self-repair analyzer (CRESTA) for embedded DRAMs," in *Proc. IEEE Int. Test Conf.*, 2000, pp. 567–574.

[38] P. Ohler, S. Hellebrand, and H. Wunderlich, "An integrated built-in test and repair approach for memories with 2D redundancy," in *Proc. IEEE Eur. Test Symp.*, 2007, pp. 91–96.

[39] M. Lee, L.-M. Denq, and C.-W. Wu, “A memory built-in self-repair scheme based on configurable spares,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 6, pp. 919–929, Jun. 2011.

[40] M.-H. Yang, H. Cho, W. Kang, and S. Kang, “EOF: Efficient built-in redundancy analysis methodology with optimal repair rate,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 7, pp. 1130–1135, Jul. 2010.

[41] C. H. Stapper, “Yield model for fault clusters within integrated circuits,” *IBM J. Res. Develop.*, vol. 28, no. 5, pp. 636–640, Sep. 1984.

Minjie Lv received the BS and PhD degrees from the Department of Electrical Engineering, Xi’an Jiaotong University, Xi’an, China, in 2010 and 2019, respectively. He was a visiting PhD student with Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, New York from 2015 to 2016. Currently, he is an associate professor with the School of Cyber Engineering, Xidian University, Xi’an, Shaanxi, China. His current research interests include high-performance computer architecture and high-reliability and low-power memory system design.

Hongbin Sun (Senior Member, IEEE) received the BS and PhD degrees in electrical engineering from Xi’an Jiaotong University, Xi’an, China, in 2003 and 2009, respectively. He was a visiting PhD student with Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, New York from 2007 to 2008. He was a post-doc with Computer Science Department, Xi’an Jiaotong University from 2009 to 2011. Currently, he is a professor with the School of Electronic and Information Engineering, Xi’an Jiaotong University. His current research interests include memory hierarchy in computer system, VLSI architecture for video processing and computer vision, and signal processing system for new memory technology.

Jingmin Xin (Senior Member, IEEE) received the BE degree in information and control engineering from Xi’an Jiaotong University, Xi’an, China, in 1988, and the MS and PhD degrees in electrical engineering from Keio University, Yokohama, Japan, in 1993 and 1996, respectively. Since 2007, he has been a professor at Xi’an Jiaotong University. His research interests include the areas of adaptive filtering, statistical and array signal processing, system identification, and pattern recognition.

Nanning Zheng (Fellow, IEEE) received the graduate degree from the Department of Electrical Engineering, Xi’an Jiaotong University, Xi’an, China, in 1975, the MS degree in information and control engineering from Xi’an Jiaotong University, Xi’an, China, in 1981, and the PhD degree in electrical engineering from Keio University, Yokohama, Japan, in 1985. He jointed Xi’an Jiaotong University, in 1975, and is currently a professor and the director of the Institute of Artificial Intelligence and Robotics, Xi’an Jiaotong University. His research interests include computer vision, pattern recognition and image processing, and hardware implementation of intelligent systems.