Redundancy Analysis using Genetic Algorithm

Helik Kanti Thacker[1], Atishay Kumar[1], Ankit Gupta[1], Keerthi Kiran Jagannathachar[1], Deokgu Yoon[2]
DRAM Solutions[1] , DRAM Product Engineering Team[2]
Samsung Semiconductor India Research and Development[1] , Samsung Electronics[2]
{h.thacker, atishay.1, ankit.g2, keerthi.k, deokgu.yoon}@samsung.com

***Abstract*—During the manufacturing of a DRAM chip, external impurities, faulty deposition steps, or manufacturing errors could generate chips with faulty memory cells rendering the chip unusable. To overcome these faulty memory cells, redundancies are included in the memory, allowing mapping of faulty rows and columns to these redundancies. The process of mapping faulty lines to redundancies is called Redundancy Analysis. Redundancy Analysis is an NP-complete problem. In this paper, we propose a memory repair solution based on the Genetic Algorithm to repair the memory efficiently without compromising on the yield compared to that of the existing heuristic algorithms. Performance comparison to the best heuristic and an exhaustive search algorithm gave a promising result with an average repair rate improvement of 6.48% and theoretical run time improvement of 33 times respectively. Genetic Algorithm can be used directly in the production line to improve the wafer yield. A compound algorithm was also developed in which the population initialization was done with the solution of a heuristic algorithm with a yield improvement of 0.5% over the genetic algorithm with random initialization.**

Keywords—redundancy analysis algorithm, NP-complete, evolutionary algorithms, genetic algorithm.

I. INTRODUCTION

Moore's Law states that the number of transistors in dense integrated circuit doubles every two years. This law also applies to increase in densities of memory, which results in an increase in the defect probability in the memory. This reduces the yield of the wafer on which the memory is manufactured. To compensate for this decrease in yield, manufacturers include redundancies in the form of spare rows and columns in memory chips so that these chips can be repaired. These spare rows and columns are mapped to faulty lines. When these faulty line addresses are requested, the mapped spare lines are accessed internally. This process of mapping spares to faulty lines is called Redundancy Analysis (RA). The chip can be used only if all the faults in the memory are repaired, otherwise the chip is discarded. Hence, the development of an efficient RA algorithm is crucial in the semiconductor industry. Achieving a very high wafer yield, while also having a feasible runtime is challenging.

The complete repair process of RA is illustrated in Fig. 1 on an 8x16 chip containing 5 faults. 1 spare row and 2 spare columns are available for allocation, using which the chip has to be repaired. The spare row is mapped to the 5th row thereby repairing 2 faults. The 2 spare columns are mapped to the 2nd and 14th columns. All the faults in the chip are repaired and hence this allocation of spares is a solution to the problem. In this case, there is only 1 way to repair the chip. In general, multiple solutions to repair the chip exist. Also, some chips are unrepairable. The main aim of RA is to repair a high number defective chips within a feasible runtime.

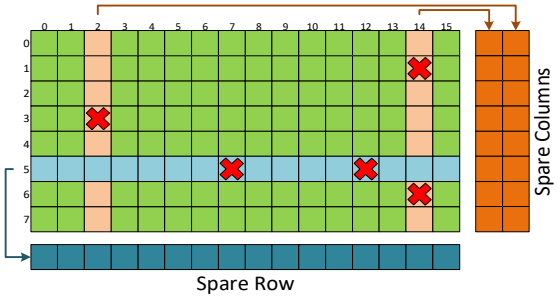

Fig. 1: Chip repair process of 5 faults using redundancies

Different RA algorithms have been proposed in literature. Since, RA is an NP-complete problem [1], only exponential algorithms will provide the highest possible yield. These algorithms are associated with complete search and are able to repair all the chips that are theoretically repairable. However, with an increase in the number of faults and chip size, the time required to repair the chips increases exponentially making them impractical to be used in the manufacturing line. On the other hand, there are various heuristic algorithms which perform in polynomial time with lower repair rates. Genetic algorithm [2] (GA) in [3], is shown as a possible RA algorithm. GA has a time complexity independent of the number of faults, for a chip size.

The summary of our contributions is as follows:

- We detail our implementation of the Genetic Algorithm as an RA algorithm
- We introduce a different population initialization for RA
- We show the theoretical scalability of our algorithm to chips of bigger sizes

In Section II, description about the existing RA algorithms and the must repair condition has been provided. The simple Genetic Algorithm has also been described, to make it easier to understand the implementation described in Section III. The experimental setup and results have been described in Section IV, followed by future work and conclusion in Section V.

II. BACKGROUND

A. Redundancy Analysis Algorithms

An efficient RA algorithm maintains a high repair rate and completes the repair in a reasonable time. Many heuristic and exhaustive RA algorithms are discussed in literature. For exhaustive RA algorithms, a full decision tree can be built by considering for each fault, all possible cases and finding a repair solution whenever it exists. The algorithm will have an exponential worst case time complexity in order of O(2n), where n is number of faults. In [4], Faulty Line Covering algorithm which is an improvement over Fault-Driven Comprehensive algorithm [5] is proposed.

978-1-6654-4175-9/21/$31.00 © 2021 IEEE

Using heuristic algorithms will produce a solution quickly but will sacrifice on the repair rate. The broadside algorithm [6] is one such heuristic algorithm which uses a greedy approach to repair the chip. It assigns spare rows or columns based on whichever is available in a greater quantity at that point in repair. LECA [4] uses Effective Coefficients to find a repair order. OSP [7] defines pivot, intersection and one side pivot faults to repair the chip.

Before applying any repair algorithm, must repair analysis [1] is generally done to reduce search space. If there are more faulty cells in a row than available spare columns, the faulty row must be repaired by a spare row. Similar condition holds for faulty columns to be in the must repair condition. In Fig. 2, column 14 with faults {E1, E8} cannot be repaired by only 1 available spare row and hence a spare column must be used to repair the column. Similarly, for row 5 with faults {E5, E6, E7}, a spare row must be used for repair since 3 faults in a row cannot be repaired with 2 spare columns.

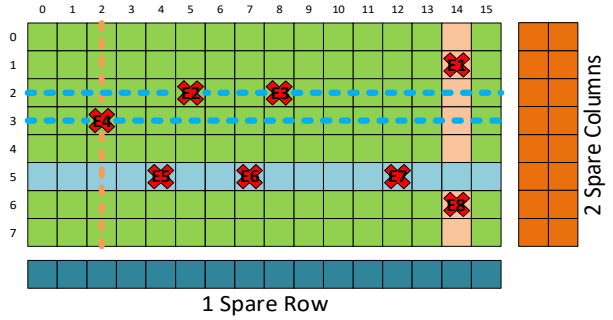

Fig. 2: Must repair with 1 spare row and 2 spare columns

B. *Genetic Algorithm*

Genetic Algorithm (GA) [2] is a heuristic evolutionary algorithm used to solve optimization problems. GA involves simulation of a population over several generations, where the population represents a collection of individuals, with each individual being a candidate solution to the problem. The initial population is usually generated randomly. The number of individuals in a generation is the population size. A fitness function is an objective function that defines how close the individual is to the final solution. Fitter individuals have a better chance to yield good solutions to the problem.

The fitter individuals take part in selection process and participate in crossover. Hence, these candidates survive by creating new offspring. Crossover is a process of recombining selected individuals to produce a new generation of individuals. A crossover can be performed by using different strategies such as N-point crossover, genes are taken alternatively between crossover points. Uniform crossover is carried out by choosing gene from parent using fitness probability. New generation individual can undergo mutation process which adds unique characteristics to individuals by randomly altering them. This process increases the search space and thus allows the solution to come out of local optima. A fixed percent of the fittest individuals are selected as elite and are given a chance to survive. These elite individuals pass to the next generation without undergoing any mutation. This ensures good solutions are not lost in the process of creating new generations.

In Section III, the implementation details of individuals, population initialization, fitness function, selection, crossover mutation, and reserving elites process of Genetic algorithm for Redundancy Analysis is explained.

III. Implementation

To solve the Redundancy Analysis (RA) problem using Genetic Algorithm (GA), an individual composed of 2 chromosomes is defined. The GA initializes the first generation of individuals and then uses these individuals to calculate fitness, and performs crossover and mutation to get a new generation of individuals. This process is carried out for a fixed number of generations.

The first step is the initialization of population where a group of individuals are defined. Fig. 3 shows 4 individuals i_1, i_2, i_3 and i_4. Multiple ways of initialization have been discussed in the paper. In the second step, the fitness of these initialized individuals is calculated. The fitness values of the individuals are calculated based on the fitness function which is discussed subsequently. In this example, individual i_1 has the highest fitness value and i_3 has the lowest. Based on the fitness, the selection of individuals into parent classes, p_1 and p_2, is performed, which will then undergo crossover. A roulette-wheel selection method is discussed in the paper.

The selection is followed by reservation of elites [8], where the fittest individuals are carried to the next generation without any changes. This is followed by the crossover and mutation steps which give a new generation of the population. In this example, individual i_1 has been reserved as an elite.

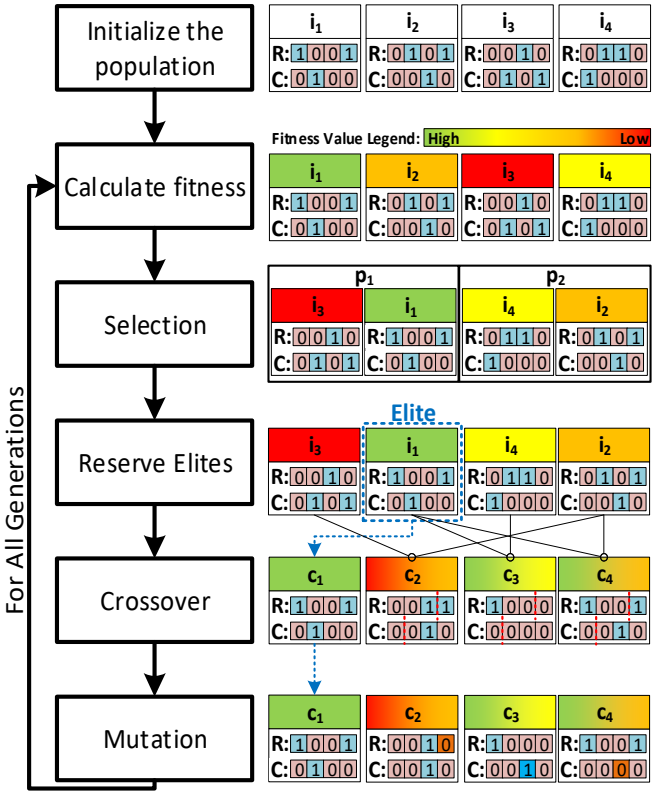

Fig. 3: Flow diagram of Genetic Algorithm implementation

Single-point crossover of parents has been illustrated in Fig. 3, where the row crossover has been performed with three genes of parent p_1 and one gene of parent p_2. Similarly for columns, crossover has been performed with one gene of parent p_1 and three genes of parent p_2. It is followed by mutation, where in this example, at max one row or column gene has been mutated from each individual, randomly. The genes marked in orange are the ones that have undergone mutation in children c2 and c4. Each of these steps have been explained in detail in individual subsections in this section.

A. Individual

An individual is the smallest representation of a solution for the Redundancy Analysis (RA) problem. As illustrated in Fig. 4, an individual consists of 2 chromosomes and a fitness value. The two chromosomes represent the spare row and column allocation in defective chips.

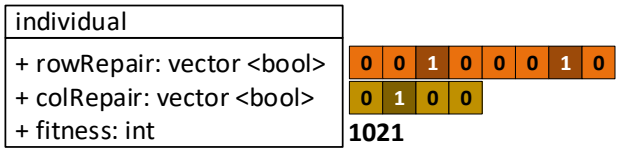

Fig. 4: Implementation of an individual

As illustrated in Fig. 5, bit strings are used to represent row and column chromosomes, where the gene value of 1 represents the allocation of spare. An 8x4 chip with one spare row and two spare columns has been used as an example. All the faults except *E4* at (2, 5) have been repaired. Here the row and column chromosomes are "0100" and "00100010" respectively indicating repairs at row 1 and columns 2 and 6.

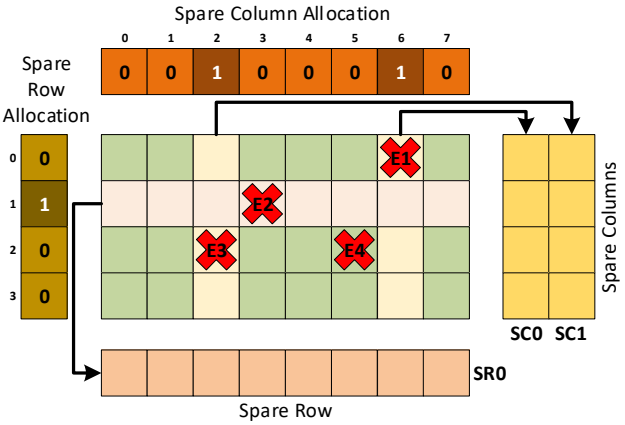

Fig. 5: Bit strings as chromosomes for memory repair

B. Population Initialization

The population initialization for the RA problem is not completely random unlike other problems which are solved using Genetic Algorithm. The first step of the initialization is the Must-Repair condition explained in Section II. This vastly reduces the number of generations required to arrive to a solution by simplifying the problem. This is followed by a probability based initialization of the first generation of individuals. This initialization method can be improved by using solutions of a fast heuristic algorithm to initialize an individual or a part of the population.

The initialization uses probability values to determine if a spare will be assigned or not. P_{Sx} can be the probability of allocation of a spare row P_{Sr} or a column P_{Sc} and is defined as:

$$P_{Sx} = P_{min} + \frac{M_f \times F_x}{M_{mf} \times F_{x\_max}}, where \quad x \in \{r, c\},\ M_{mf} > M_f\ and\ P_{min} + M_f/M_{mf} \leq 1 \qquad (1)$$

where P_{Sr} can be calculated using P_{min}: minimum probability of assigning a spare to a faulty row, F_r: number of faults in that row, F_{r_max}: maximum number of faults in any row, and M_f and M_{mf}: the fault multipliers. So, if a P_{Sr} value between 0.4 and 0.6 is desired, the P_{min} would be 0.4, and the M_f and M_{mf} values can be 1 and 5, so the second term on the R.H.S of (1) has a probability range [0, 0.2] restricting the range of P_{Sr} in [0.4, 0.6]. Similar calculations are carried out for P_{Sc}.

C. Fitness function

The fitness function determines the ability of an individual to perform a quality repair. The fitness function is divided into 3 parts for calculations, which are based on linear fault solving score, multi-repair reward and repair penalization. The fitness function was initially implemented only with the linear fault solving but it was then improved by adding multi-repair reward and repair penalization to the function.

Linear Fault Solving Score: The linear fault solving score of an individual depends on the number of faults that have been solved and the number of spares that have been used. The linear fault solving function, returns a higher value if the number of faults solved are more and the number of spares used are less. The linear fault solving fitness (f_{lfs}) is defined as:

$$f_{lfs} = \alpha \cdot fault_{solved} - \beta \cdot ns_r - \gamma \cdot ns_c \qquad (2)$$

Where α, β and γ are the positive multiplication constants to change the weight given to the different factors, $fault_{solved}$ is the number of faults solved by the individual, and ns_r and ns_c are the number of spare rows and columns used by the individual respectively. 2 individuals with different solutions using the same number of spares is illustrated in Fig. 6. The individual in Fig. 6 (b) has a higher f_{lfs} value as it has repaired 3 faults in the chip as compared to 2 faults repaired by 6 (a). Since 6 (b) has a higher fitness value, it will have a better chance of producing an individual which is able to repair all the faults present in the chip.

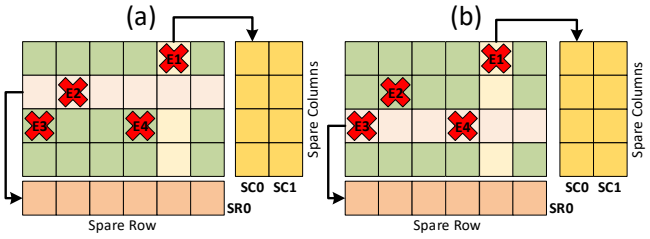

Fig. 6: Linear Fault Solving example

Multi-Repair Reward: An individual is awarded with a multi-repair reward if it is able to find solutions repairing multiple faults with a single spare row or column. The repair threshold is decided beforehand. If the number of faults repaired exceeds that threshold, a multi-repair reward directly proportional to the number of faults is awarded to the individual based on the following equation:

$$f_{mrr} = \delta \cdot (\delta_r \sum_{i \in \{R_r\}} f_i \cdot \mathbf{1}(f_i \geq ft_r) + \delta_c \sum_{j \in \{R_c\}} f_j \cdot \mathbf{1}(f_j \geq ft_c) \quad (3)$$

Where δ is the overall multiplication factor of the multi-repair reward, δ_r and δ_c are the multiplication constants for row and column multi-repair respectively. R_r is the set of row repairs performed by the individual whereas R_c is the set of column repairs. **1** is the indicator function. In Fig. 7, the number of faults repaired by a spare row is checked for the reward. In this case, rows 0 and 5 are checked. Assuming that the fault threshold for row, $f_{tr} = 3$. For row 0, $f_0 < f_{tr}$, so the individual is not rewarded for that, but for row 5, $f_5 \geq f_{tr}$, thus it'll be rewarded proportional to f_5 which is $3 \cdot \delta_r$. Similarly in the case of columns, assuming the fault threshold for column, $f_{tc} = 3$, the individual will be rewarded for repairing columns 10 and 14 because $f_{10} >= f_{tc}$ and $f_{14} >= f_{tc}$ but not for column 2 since $f_2 < f_{tc}$.

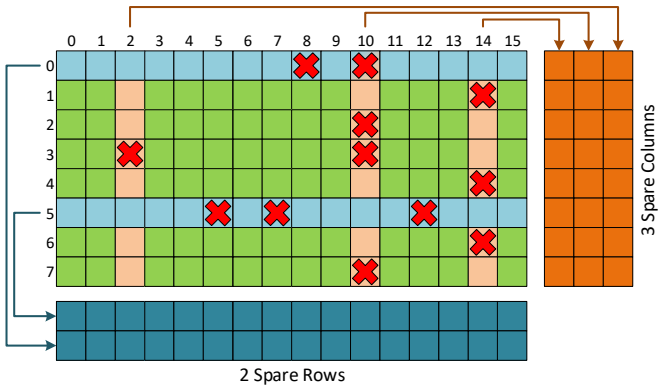

Fig. 7: Multi-repair Reward example and Repair penalization example

Repair penalization: When faults are shared between rows and columns, it is possible for the same fault to be repaired with both a spare row and a column. For example in Fig. 7, the fault at (0, 10) has been repaired with both a spare row and a column. In case an individual decides to repair row 0 first and not repair column 10 with a spare column, it should be penalized as the fault at (0, 10) is more suited to a column repair. That is why, we penalize an individual if a lot of faults are repaired both by both a spare row and a spare column. The penalty is proportional to the number of such faults.

This penalization (f_{rp}) is based on the following formula:

$$f_{rp} = \lambda \times \begin{cases} \sum_{i \in \{R_r, R_c\}} \sum_{j \in \{i_c, i_r\}} f_j, & f_j \geq f_t \\ 0, & f_j < f_t \end{cases} \quad (4)$$

Where λ is the repair penalization multiplier, and for all the rows and columns that have been repaired ($i \in \{R_r, R_c\}$), all the faults are iterated over ($j \in \{i_c, i_r\}$), and the fault count, f_j of that row or column is the penalty for repair of that fault. If f_j exceeds the fault threshold f_t, only then the individual is penalized. For example in Fig. 7, if row 0 is repaired, since the fault (0, 8) is the lone fault in column 8, it is not penalized but fault (0, 10) has 3 more faults in column 10. Assuming this crosses the fault threshold value, it will be penalized by $4 \cdot \lambda$ points as the column count for column 10 is 4. In case of columns, Table I shows the penalization of different columns that have been repaired in Fig. 7. Here the assumed fault threshold value, f_t is 2 and λ is 1.

TABLE I. COLUMN REPAIR PENALIZATION EXAMPLE

Col	Count	Faults (x, y) and f_j for (x, y)				f_{rp}
2	1	(3, 2)	-	-	-	2
	i=2, f_j	2	-	-	-	
10	4	(0, 10)	(2, 10)	(3, 10)	(7, 10)	4
	i=10, f_j	2	0	2	0	
14	3	(1, 14)	(4, 14)	(6, 14)	-	0
	i=14, f_j	0	0	0	-	

The fitness function is then calculated using (2), (3) and (4) with the help of the following equation:

$$\text{fitness} = \begin{cases} -1 \times k, & n_{sr} > T_{sr} \text{ or } n_{sc} > T_{sc} \\ k + f_{lfs} + f_{mrr} - f_{rp}, & else \end{cases} \quad (5)$$

where k is a constant used to mark illegal solutions using more spare rows and columns than available. So, in case of a legal solution, fitness is the sum of k, linear fault solving score and the multi-repair reward, with the repair penalization subtracted from it.

Hypothesis 1. The proposed fitness function differentiates between solutions of a common complex case in DRAM chips.

Assumptions.

1. $|SR| = n_R$
   - if $|SR| < n_R$, not solvable if any(n_{Rei}) > $|SC|$)
   - if $|SR| > n_R$, easily solvable
2. $n_{Ctotal} \leq |SC|$, if $|SC| > n_{Ctotal}$, not solvable
3. $n_C >= |SC|$, if $n_C < |SC|$, solvable
4. $n_{Ctotal} < f_{tr} < n_R$
5. $n_C < f_{tc}$

where |SR| and |SC| are the number of spare rows and columns allocated, n_R and n_C are the number of shared rows and columns. n_{RC} equals $n_R \cdot n_C$, and f_{tr} and f_{tc} are the row and column fault thresholds. |F| is the number of faults in the chip.

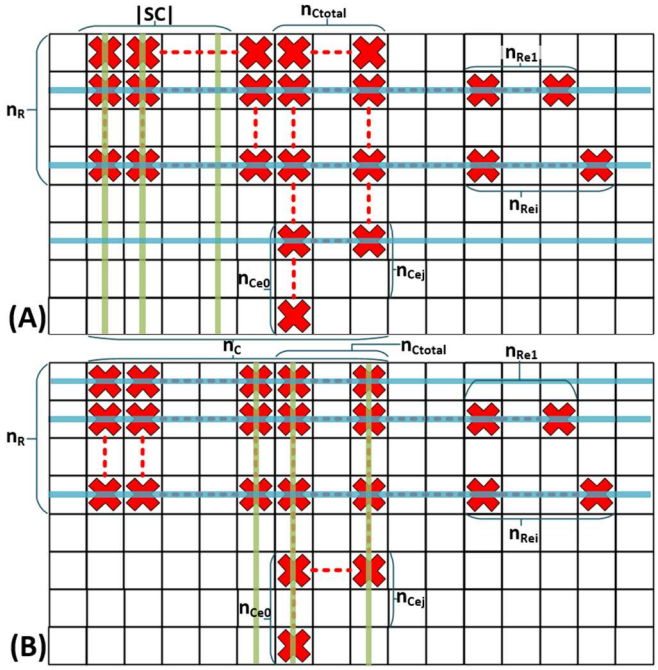

Fig. 8: General Case of DRAM Chip

Proof. We consider a common case and show that the fitness function will accurately differentiate the two solutions.

Fitness function calculation for Fig. 8(A):

$$f_{lfs} = \alpha.\left[|F| - \left(\sum_{j=0}^{n_{Ctotal}} n_{Cej} + n_{RC} - |SC|\right) - \beta.|SR| - \gamma.|SC|\right]$$

$$f_{mrr} = \delta_R.\left(|SR|.n_{RC} + \sum_{i=0}^{|SR|} n_{Rei}\right) + \delta_c.0$$

$$f_{rp} = \lambda.(|SC|.|SR|)$$

Fitness function calculation for Figure 8(B),

$$f_{lfs} = \alpha.|F| - \beta.|SR| - \gamma.\ n_{Ctotal}$$

$$f_{mrr} = \delta_R.\left(|SR|.n_{RC} + \sum_{i=0}^{|SR|} n_{Rei}\right) + \delta_C.z\ , where$$

$$z\ \epsilon\ \left[0, \left(n_{Ctotal}.|SR| + \sum_{j=0}^{n_{Ctotal}} n_{Cej}\right)\right]$$

$$f_{rp} = \lambda.(n_{Ctotal}.|SR|)$$

Taking the difference, we get,

$$f_{lfs_{B-A}} = \alpha.\left[|F| - |F| + \left(\sum_{j=0}^{n_{Ctotal}} n_{cej} + n_{RC} - |SC|\right)\right] - \gamma.(n_{Ctotal} - |SC|) > 0$$

$$f_{mrr_{B-A}} = \delta_R.(n_{Re0}) + \delta_c.\left(n_{Ctotal}.|SR| + \sum_{j=0}^{n_{Ctotal}} n_{Cej}\right) > 0$$

$$f_{rp_{B-A}} = \lambda.|SR|.(n_{Ctotal} - |SR|) < 0$$

$$\therefore Fitness_B - Fitness_A = f_{lfs_{B-A}} + f_{mrr_{B-A}} - f_{rp_{B-A}} > 0$$

$$\therefore Fitness_B > Fitness_A$$

D. Selection

A roulette-wheel selection [9] is followed where k individuals are chosen to be included in the set p_1 and another k individuals in p_2. The selection process starts by sorting the individuals based on their fitness values. This is followed by rank based selection as shown in Fig. 9, where individuals with a higher rank have a higher probability of selection. In the example, with population size $p = 8$, the individual with the first rank has a probability of selection equal to 2/9 and the individual with rank 8 has a probability of selection equal to 1/36. This probability for rank r is calculated by the following equation:

$$p(r) = 2 \times (p - r + 1)/(p \times (p + 1))$$

After this, the probabilities are assigned and individuals are selected into sets p_1 and p_2 for crossover.

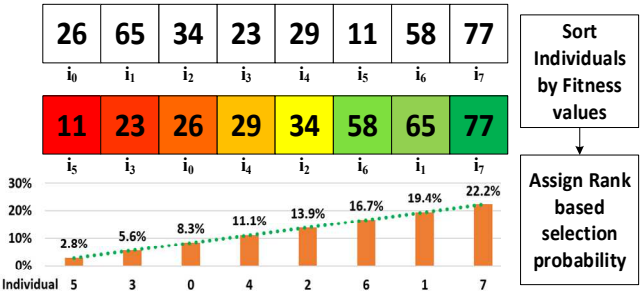

Fig. 9: Selection process in Genetic Algorithm. The values in the boxes represent the fitness value of the individuals

E. Crossover

Single-point crossover is implemented for individuals selected into the 2 sets p_1 and p_2 in the previous step. The crossover point is independently selected for both row and column bit strings. In Fig. 10, two individuals, Individual 1 and 2 are crossed over. Each of the individuals has 2 chromosomes, one for spare row and the other for spare column allocation. These chromosomes are crossed over independently using a crossover point of 3rd and 1st gene respectively. These crossed over genes are passed to the intermediate individuals 1-2 and 2-1 which will go for mutation in the next step.

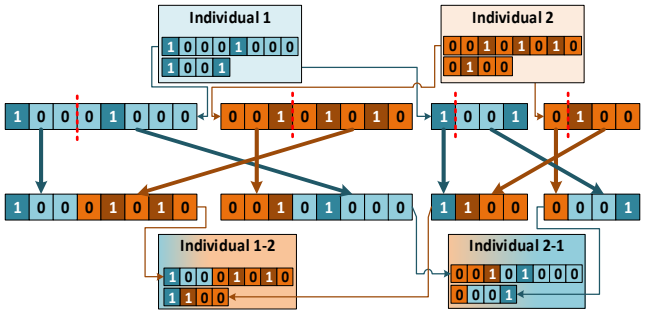

Fig. 10: Crossover example with 2 individuals

F. Mutation

The mutation operation [2] consists of flipping the value at a particular bit position with a probability p. Mutation probability p is kept slightly higher to increase the search space. This helps in avoiding getting stuck in local optimum. Mutation is carried out in 2 steps for each chromosome. Initially, the genes with value of 1 in the chromosome are mutated to zeros independently with a probability of p_{10}. Then the spare count is updated as some extra spares will become available. Then, mutation is done from zeros to ones with a probability of p_{01}, till spares are remaining or the mutation process is over.

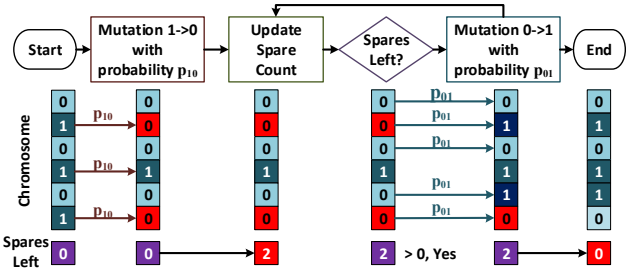

Fig. 11: Mutation for an individual

In Fig. 11, a 6 bit chromosome with 3 spares is illustrated. First, 1 to 0 mutations occur with a probability of p_{10} which mutates 2 out of 3 1s to 0s in the chromosome. Then the spare count is updated to 2 allowing 0 to 1 mutations. We see 2 out of 5 possible 0 to 1 mutations giving us the final chromosome.

G. Reserving elites

As an additional step, to ensure that the best individual of the current generation is not worse than the previous generations, a subset of the current population called elites is guaranteed a place in the next generation [8]. Elites are selected by sorting the individuals based on their fitness values and picking the top n individuals with the highest fitness.

IV. Experimental Setup and Results

A. Execution Time Comparison

Fig. 12 illustrates the theoretical comparison of number of instructions required by FLCA and our Genetic Algorithm in 10^9 instructions. The Genetic Algorithm starts to require lesser instructions than FLCA at 210, 228 and 250 number of faults for 2, 4 and 16 Gb DRAM chips respectively.

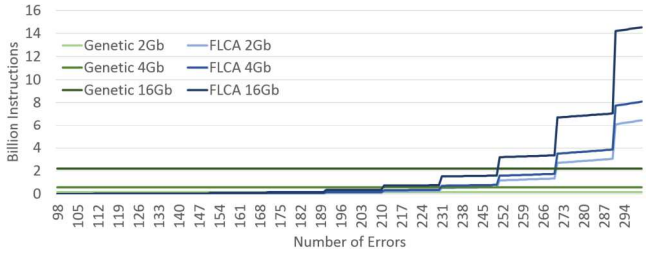

Fig. 12: Theoretical comparison of FLCA and Genetic algorithm

B. Repair rate

We compare the performance of the Genetic Algorithm (GA) with other algorithms in Fig. 13, on 1.3×10^7 simulated chips of size 64×64 with 8 spare rows and columns. With an increase in fault rates, we observed a sharp decrease in the existing heuristic algorithm yield after a fault rate of ~1.1%. However, the Genetic Algorithm's yield was within 2% of the exponential algorithm FLCA. Thus, GA was able to maintain a very high repair rate of 98.2% even at high chip fault rates of ~1.6%. GA solves 1.97% unique chips compared to the existing heuristic algorithms.

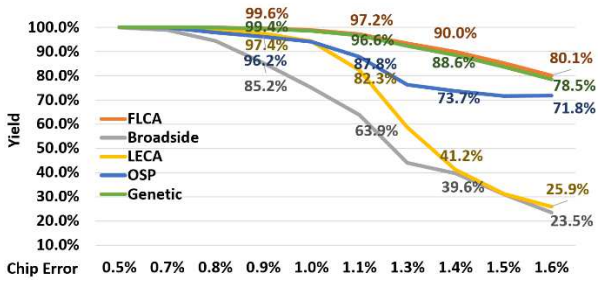

Fig. 13: Yield comparison of GA with existing RA algorithms

The parameters used for the GA are shown in Table II.

TABLE II. Parameters Used For The Genetic Algorithm

Parameter	Explanation	Value
Generations	#iterations of the GA	1000
Population	#individuals in a generation	300
Elites	#elites reserved per generation	10
P_{min}, M_f, M_{mf}	constant, fault multiplier in (1)	0.6, 1, 5
p_{10}, p_{01}	1-0 and 0-1 mutation probability	0.1, 0.2
α, β, γ	constant used in (2)	20, 10, 10
δ, λ	multiplier used in (3) and (4)	1, 1
f_{tr}, f_{tc}	row, column threshold in (3)	3, 3
f_t	fault threshold used in (4)	3
k	constant used in (5)	1000

C. Population Fitness of Genetic Algorithm

Fig. 14 illustrates the population quality of the Genetic Algorithm over the generations. Each strip represents a single generation. The color gradient of the general population varies between red and green, with green being the best fitness value for all individuals in all generations and red, the worst. Black represents an invalid solution when more spares are used than available. Blue denotes individuals that have repaired the chip completely. Over the generations, a quality improvement can be seen as the invalid and the inferior solutions i.e., orange, red and black, are decreasing and the superior solutions i.e., green and blue, are increasing.

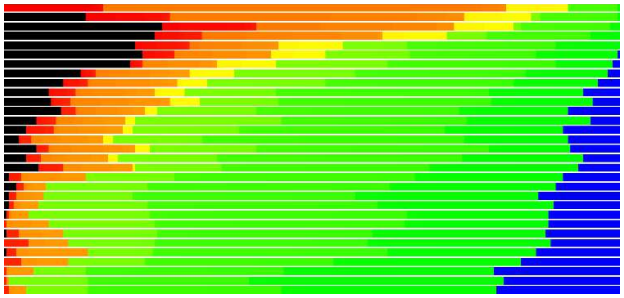

Fig. 14: Population quality of subsequent Generations

D. Reserving Elites and Compound Initialization

Reserving the elites improved the yield of our genetic algorithm by 2%. Additionally, initializing a small part of the initial population with the solution of the Broadside algorithm and its mutations further improved the yield by 0.5%. Since the Broadside algorithm has a very low runtime, it is feasible to use it to initialize part of the initial population.

V. Future Scope and Conclusion

Each individual in a generation is independent of the other, so the algorithm is also being implemented using parallel programming in CUDA to bring down the execution time. Without the use of exponential algorithms, it is difficult to achieve a 100% normalized repair rate. The experiments have shown promising results when the Genetic Algorithm is used even at high fault rates. With decreasing node sizes and increasing densities in upcoming DRAM technologies like DDR5 and LPDDR5, the FLCA algorithm will not be able to repair the chips after a certain number of faults because of its exponential time complexity. Thus, the Genetic Algorithm is a potential candidate to be used in Redundancy Analysis.

References

[1] S. K. Cho, K. Wooheon, C. Hyungjun, et al., "A Survey of Repair Analysis Algorithms for Memories", ACM Computer Survey, 2016.

[2] John H. Holland, "Adaptation in Natural and Artificial Systems". Ann Arbor, MI: University of Michigan Press, 1975.

[3] J. Milbourn, "Strategies for Optimising DRAM Repair," Durham theses, Durham University, 2010.

[4] F. Lombardi, et al., "Approaches for the repair of VLSI/WSI RRAMs by row/column deletion," ICFTCS, 1988, pp. 342-347

[5] J. R. Day, "A fault-driven comprehensive redundancy algorithm," IEEE Design & Test, pp. 35–44, Jun. 1985.

[6] M. Tarr, et al., "Defect Analysis System Speeds Test and Repair of Redundant Memories." Electronics. pp. 175-179. 1984.

[7] J. Kim, et al., "A new redundancy analysis algorithm using one side pivot," International SoC Design Conference, pp. 134-135, Jeju, 2014.

[8] E. Zitzler, et al., "Comparison of multiobjective evolutionary algorithms: Empirical results". Evolutionary Computation, 8 (in press).

[9] D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning, 1st ed. Addison-Wesley, 1989