EMGA: An Evolutionary Memory Grouping Algorithm for MBIST

Yang Li[1], Yongqiang Duan[1], Hao Zhang[1], Dan Niu[2], Xiao Wu[3] and Zhou Jin[1]
1.Super Scientific Software Laboratory, China University of Petroleum-Beijing, Beijing, China
2.School of Automation, Southeast University, Nanjing, China
3.Huada Empyrean Software Co. Ltd, Beijing, China
Email: liyang@student.cup.edu.cn, duanyongqiang@student.cup.edu.cn, zhanghao@student.cup.edu.cn, 101011786@seu.edu.cn, wuxiao@mail.empyrean.com.cn, jinzhou@cup.edu.cn

***Abstract*—MBIST (Memory Built-In Self-Test) is a widely used methodology in chip design and fabrication to detect and localize faults in memories. Due to the large memory sizes of modern chips, memories need to be grouped in order to manage and test them efficiently. However, due to the high number of constraints and memories, the time complexity of solving directly using heuristic algorithms is high and the grouping results obtained are of poor quality. In this paper, we propose a heuristic-based MBIST grouping algorithm to maintain high efficiency while achieving high quality grouping. We firstly divide the numerous constraints into two categories to reduce the constraint dimensions and obtain an initial grouping result. We then use a greedy algorithm with a penalty term to quickly obtain the result that satisfies all the constraints from the initial result in order to reduce the time consumption and the size of the grouping. In order to avoid local optimal solutions, we further use an improved genetic algorithm to optimize the result of the greedy algorithm to obtain higher quality groupings. The experimental results demonstrate that our algorithm reduces the number of groups 119.44% on average compared with the K-Means method. Compared with simulated annealing algorithm and genetic algorithm, EMGA reduces the number of groups by 8.35% and 4.66%, and time by 79.51% and 73.30%, respectively.**

***Index Terms*—MBIST, grouping, greedy algorithm, genetic algorithm**

I. Introduction

The design for testability (DFT) in digital circuit design refers to the ability to consider test and fault localization during the chip design process [1]. As chip complexity increases and manufacturing processes advance, traditional test methods can no longer meet the demand for efficient and accurate testing. Therefore, DFT has become the key to resolving this problem [2]. Reasonable testability design can improve the quality and reliability of the chip.

MBIST (Memory Built-In Self-Test) is a technique in DFT used for memory testing, which realizes self-testing by embedding test circuits in the design of integrated circuits [3]. By adding specialized test circuits and control logic inside the memory, MBIST technology allows the memory to automatically execute test patterns and perform write, read, and compare operations on the memory cell, thereby detecting faults in the memory cell. MBIST technology can be tested during the chip manufacturing process, or while the chip is in operation, in order to ensure the reliability and correctness of the memory [4]. It has become a commonly used technique in modern integrated circuit design and is particularly suitable for testing and fault diagnosis of large-scale memories [5].

Due to the large number of memories in the chip, testing the whole chip individually will consume a lot of time and test resources. Therefore, memory grouping allows the entire test process to be decomposed into multiple smaller test tasks, each focusing only on a specific memory group. This reduces the size and complexity of testing and improves test parallelism and efficiency. At the same time, memory grouping can also help discover and locate faults in memory groups, facilitating fault diagnosis and repair [6].

Current work focuses more on exploring efficient testing schemes, memory fault modeling, and improvement of fault localization techniques. There are also some scholars who use machine learning to perform memory grouping. Machine learning algorithms usually require large amounts of training data to achieve good performance [7]. The data collection and labeling process can be very time-consuming and laborious. And if the training data is insufficient or inaccurate, it may lead to degraded model performance or produce unreliable grouping results. Therefore, traditional heuristic algorithms and optimization methods may still be viable and effective options [8].

In this paper, we propose an innovative evolutionary memory grouping algorithm, EMGA, to improve efficiency and get high quality groupings. In order to quickly obtain an initial input, we first classify the constraints into two categories: hard and soft constraints, and obtain the initial grouping result by hard constraints. We then use a greedy algorithm with a penalty term to quickly obtain a result that satisfies both types of constraints from the initial result in order to reduce time consumption and group size. In order to avoid local optimal solutions, we further use an improved genetic algorithm to optimize the result of the greedy algorithm to obtain higher quality groupings. We verify the performance of the proposed algorithm on 5 test sets. The experimental results demonstrate a 119.44% reduction in the number of groups in EMGA compared with the K-Means algorithm. Compared with simulated annealing algorithm and genetic algorithm, EMGA reduces the number of groups by 8.35% and 4.66% and time by 79.51% and 73.30%.

II. Problem Statement

A. MBIST Grouping

As shown in Fig. 1, efficient planning and grouping of memory cells and their assignment to different MBIST controllers is critical to optimizing the test flow and ensuring the reliability of the memory subsystem.

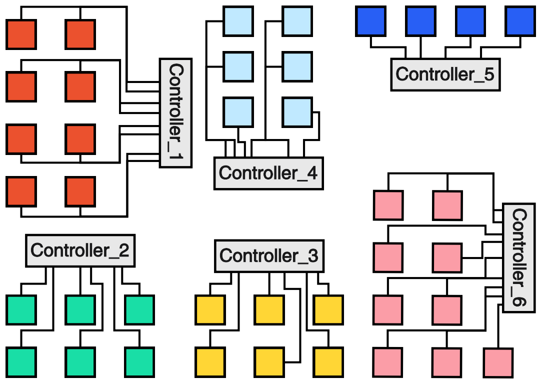


Fig. 1: Memory grouping.

The need for MBIST grouping stems from several key considerations. First, by grouping memories, phased testing can be achieved, thus reducing overall test time. More targeted testing in different test phases can improve test efficiency. Second, different memory groups may have different characteristics and test requirements. Some memory groups may be more sensitive to read and write operations, while others may be more focused on boundary condition testing. Grouping memories enables better allocation of test resources for optimal test coverage. Third, grouping helps with fault localization. If a problem is found during testing, partitioning helps to narrow down the scope of the fault so that it is easier to find the specific memory group with the problem in order to quickly locate and troubleshoot the problem.

B. Multi-Constraint Problem and Optimization

The grouping process begins with the problem of multiple constraints. Multiple constraints means that various factors or constraints need to be considered simultaneously in the grouping process. These restrictions may relate to different characteristics. Only memories that satisfy all the constraints can be grouped, and the less the final grouping result, the better. In this paper, we select some of the following constraints which are usually the most important and define them formally.

Given a set of memory sets M:

$M = \{M_1, M_2, M_3, \ldots, M_m\}$, m is the total number of memories.

The memory grouping set is G:

$G = \{G_1, G_2, G_3, \ldots, G_n\}$, n is the total number of groups.

If each group contains k memories, power consumption limit for each group of memory is:

$$\sum_{i=1}^{k} Power_i \leq Max\_Power \tag{1}$$

And $\forall M_i, M_j \in G_n$:

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq Max\_Distance \tag{2}$$

There are also some restrictions on attribute classes, $\forall M_i, M_j \in G_n$:

$$\begin{cases} M_i.\text{LogicalAddressMap} = M_j.\text{LogicalAddressMap} \\ M_i.\text{clk} = M_j.\text{clk} \\ M_i.\text{type} = M_j.\text{type} \\ M_i.\text{operationSet} = M_j.\text{operationSet} \\ M_i.\text{logicalPorts} = M_j.\text{logicalPorts} \\ M_i.\text{ShadowRead} = M_j.\text{ShadowRead} \\ M_i.\text{ShadowWrite} = M_j.\text{ShadowWrite} \\ M_i.\text{ShadowWriteOK} = M_j.\text{ShadowWriteOK} \\ M_i.\text{WriteOutOfRange} = M_j.\text{WriteOutOfRange} \end{cases} \tag{3}$$

Optimization is the primary goal, aimed at finding the best solution while considering these constraints. Faced with numerous constraints, it is unacceptable to consider the complexity of grouping simultaneously. Optimization methods need to be considered for step-by-step judgments to reduce the time complexity by reducing the constraint dimensions at each stage. In addition, the optimization process must be adaptive and able to respond to changes in constraints or objectives.

C. Genetic Algorithm

The genetic algorithm (GA) is an optimization algorithm that simulates the natural evolutionary process and is based on the principles of genetics in biology and Darwin's theory of evolution. This algorithm solves optimization problems by simulating evolutionary operations such as natural selection, crossover, and mutation to select individuals with higher fitness from an initial population [9].

GA is inherently adapted to multi-objective optimization problems and can optimize multiple objective functions simultaneously. It is able to perform global search in the entire search space and does not easily fall into local optimal solutions, which is suitable for complex multi-peak functions and high-dimensional optimization problems. GA can also easily cope with changes in the problem, such as modifying the objective function and adjusting the constraints. Due to its population-based evolutionary nature, the algorithm is relatively insensitive to changes in the problem structure and can adapt to different types of optimization problems. However, since the genetic algorithm is a stochastic search algorithm, it usually requires a lot of iterations and calculations to find a better solution. For complex problems, the search process of the algorithm may take a long time to converge.

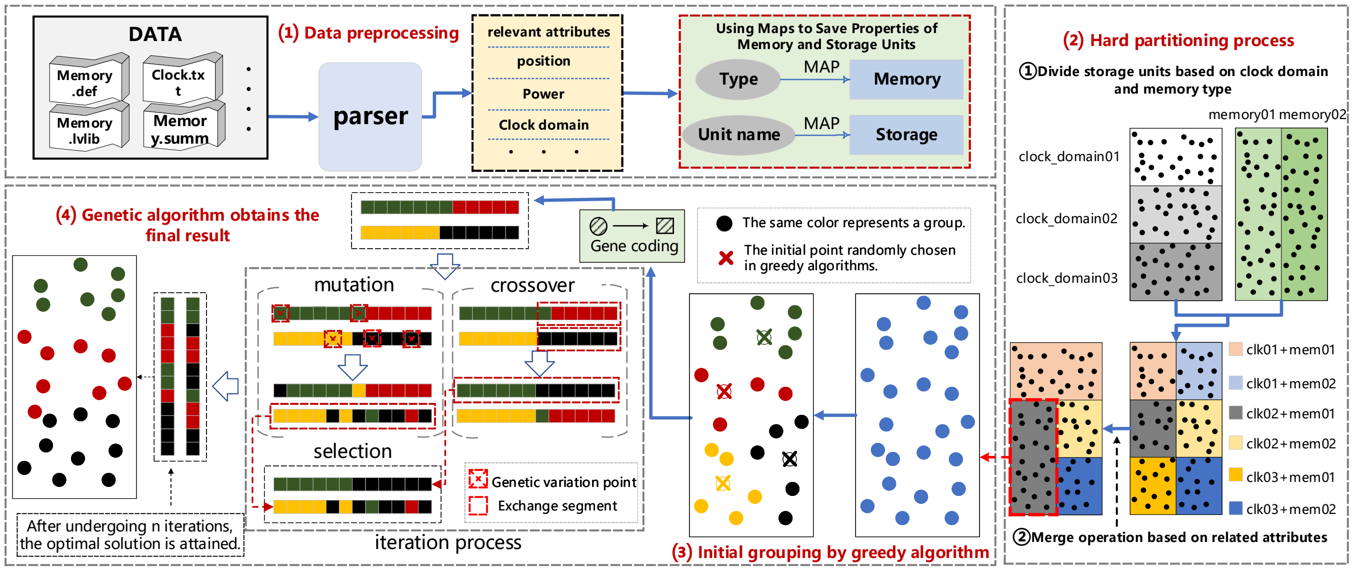

DATA
Memory .def
Clock.txt
Memory .lvlib
Memory y.summ
(1) Data preprocessing
parser
relevant attributes
position
Power
Clock domain
Using Maps to Save Properties of Memory and Storage Units
Type
MAP
Memory
Unit name
MAP
Storage
(2) Hard partitioning process
①Divide storage units based on clock domain and memory type
memory01 memory02
clock_domain01
clock_domain02
clock_domain03
clk01+mem01
clk01+mem02
clk02+mem01
clk02+mem02
clk03+mem01
clk03+mem02
②Merge operation based on related attributes
(4) Genetic algorithm obtains the final result
Gene coding
The same color represents a group.
The initial point randomly chosen in greedy algorithms.
mutation
crossover
selection
Genetic variation point
Exchange segment
iteration process
After undergoing n iterations, the optimal solution is attained.
(3) Initial grouping by greedy algorithm

Fig. 2: Overall framework of MBIST grouping.

III. Algorithm Flow

A. Overall Process

As shown in Fig. 2, EMGA contains three main parts. The first part is parser and hard partitioning. Due to the fact that attribute information such as memory type, power consumption, and various grouping constraints almost all exist in the design file of the memory, it is necessary to extract the memory information and constraints from the file and save them in the corresponding data structure. After reading the information, we can divide the constraints into hard constraints and soft constraints. As shown in Fig. 3, based on hard constraints, the memories is hard partitioned to obtain initial groupings. The second part uses a greedy algorithm to reduce the size of each grouping and quickly obtain results that satisfy both hard and soft constraints based on the initial groupings. The third part uses genetic algorithms to find fewer groupings to avoid local optimal solutions.

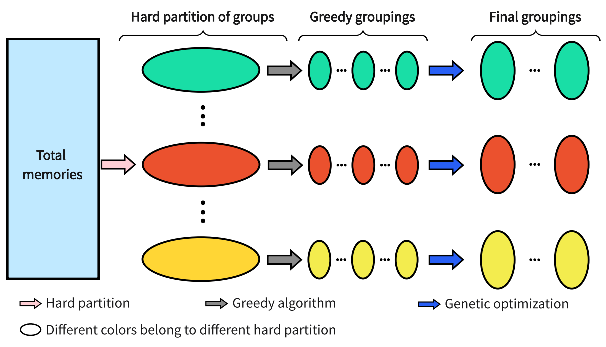


Fig. 3: The three parts of EMGA.

B. Parser and Hard Partition

The main problem encountered during the data reading process is the differences in file format and encoding methods. For example, for the power attribute of memory, it is stored in three different types of files and needs to be obtained according to priority. In the specific implementation, we first obtain the value from lvlib. If it exists, we will read the value. If it does not exist, we will read it from summ files according to priority until we obtain the value. For files with the same attributes but different formats, we adopt a unified strategy of converting them to lowercase format for consistency checking and extraction. Given the differences in file processing methods among different operating systems, we perform preprocessing before reading files to ensure smooth operation in subsequent operations, and so on.

Due to the large amount of constraint information read in, the complexity of grouping directly is unacceptable. It is necessary to consider whether grouping can be divided into multiple steps to reduce complexity. We classify constraint information into two categories. In the constraints mentioned in Section II, the condition of using an equal sign constraint can be called a hard constraint, which contains LogicalAddressMap, clk, type, operationSet, logicalPorts, ShadowRead, ShadowWrite, ShadowWriteOK and WriteOutOfRange. The conditions of inequality constraints can be referred to as soft constraints, namely power consumption limitations and distance limitations. During the parser process, an initial grouping can be quickly obtained through hard constraints.

C. Improved Greedy

The initial grouping obtained in the previous step will be the initial input to the greedy algorithm. This step handles each hard-divided grouping separately. Use the greedy algorithm to continue dividing each grouping so that it satisfies the soft constraints.

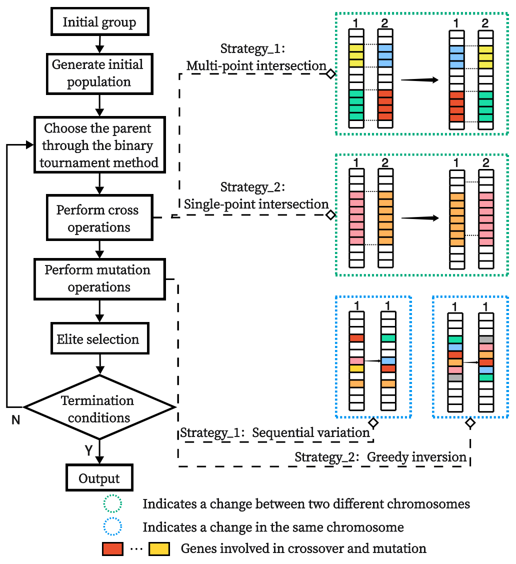
When applying greedy algorithms, it is difficult to adjust the optimization strategy using random greedy algorithms. The randomness of random greedy algorithms makes their behavior more uncertain, making it difficult to adjust the optimization strategy of the algorithm. In order to make the greedy algorithm to get a relatively good result after that, we add loss function to improve the greedy algorithm.

P_g is the number of groupings when considering only power consumption, and L_g is the number of groupings when considering only distance. λ is $\frac{P_g}{P_g+L_g}$, and μ is $\frac{L_g}{P_g+L_g}$. W_i is the power consumption after adding the i-th memory to the current group, and D_i is the maximum distance after adding the i-th memory to the current group. Max_Power and $Max\_Distance$ are the maximum power consumption and maximum distance. The loss function is defined as:

$$F = \lambda \cdot \frac{W_{i+1} - W_i}{Max\_Power} + \mu \cdot \frac{D_{i+1} - D_i}{Max\_Distance} \tag{4}$$

This loss function is mainly introduced to generate better initial solutions quickly and efficiently, and the memory with the smallest loss function value is always chosen each time. In the initial stage of greedy algorithms, the goal is to allocate data to memory as close as possible to slow down the maximum distance growth rate allocated by the controller. Due to the low memory power consumption, the loss function is temporarily considered to have lower memory power consumption. At the end of the greedy algorithm, as the distance between memory is close to the upper limit, it is attempted to add memory that meets the distance requirements and has lower power consumption to the group. This will end up with a new grouping.

D. Optimization through Genetic Algorithm



Initial group
Generate initial population
Choose the parent through the binary tournament method
Perform cross operations
Perform mutation operations
Elite selection
Termination conditions
N
Y
Output
Strategy_1: Multi-point intersection
Strategy_2: Single-point intersection
Strategy_1: Sequential variation
Strategy_2: Greedy inversion
1
2
Indicates a change between two different chromosomes
Indicates a change in the same chromosome
Genes involved in crossover and mutation

Fig. 4: Population iteration process.

A good quality initial solution obtained by the improved greedy algorithm will be used as the initial input to the genetic algorithm.

As a result of the constraint partitioning, the current constraints are left with soft constraints, i.e., power consumption and distance. The constraint on power consumption is similar to a one-dimensional packaging problem, and after adding distance constraints, this problem can be seen as a one-dimensional packing problem with conflicts. In traditional one-dimensional packing problems, genetic algorithms have the best performance. Genetic algorithms are usually divided into two main parts, gene coding and population iteration. In the genetic coding phase, we use real number coding. Compared to discrete coding, real number coding provides more solution space, which helps to find better solutions. In the population iteration phase, we employ diverse crossover and mutation techniques to increase the likelihood of jumping out of the locally optimal solution. Further, to improve the temporal performance, we adopt a space-for-time strategy and introduce adaptive mutation to optimize the quality of the population.

Algorithm 1 Genetic Algorithm

```
Require: $D = \{x_1, x_2, ..., x_n\}$, $D$ is the initial grouping
Ensure: $R = \{x_1, x_2, ..., x_m\}$, $R$ is the final grouping
 1: for $i$ in $n$ do
 2:   $x_i \xrightarrow{encoding}$ Initialize group$\{\zeta_1, \zeta_2, ..., \zeta_8\}$
 3:   while Failure to meet termination conditions do
 4:     Generate parent generation
 5:     for $j$ in 8 do
 6:       Choose $u, v$, $Min(F(\zeta_u), F(\zeta_v)) \xrightarrow{in} P$, $F(\zeta)$ is
          fitness function, $P$ is the parent generation
 7:       Update $j \leftarrow j + 1$
 8:     end for
 9:     Cross connection
        $random < probability$ ? Multi-point crossing :
        Single-point crossing
10:     Variation
        $random < probability$ ? Sequential variation :
        Greedy inversion
11:     Update $F(\zeta) \leftarrow F(\zeta)$
12:     Elitist selection
13:   end while
14:   Update $i \leftarrow i + 1$
15:   $Min(F(\zeta_1), F(\zeta_1), ..., F(\zeta_8)) \xrightarrow{decoding} R$
16: end for
17: return $R$
```

1) Gene Coding: In coding schemes, the traditional encoding method is generally binary encoding. However, considering that both the number of groups and the amount of memory are integers, real number encoding can consider the problem space as a continuous space. In this way, genetic algorithms can search in continuous space. In addition, for problems with a large number of groups or memory, using real number encoding can provide a more fine-grained representation. In binary coding, each gene locus can only take 0 or 1, leading to the discreteness and limitations of the solution space. Real number encoding can provide more range of values and selection space. So using integer encoding for chromosomes is more appropriate, where each gene segment represents a memory.

2) Population Iteration: The steps of population iteration are shown in Figure. 4. To accelerate convergence as well as to save computational resources, we use the results of the greedy algorithm as the original initial solution for the iterative process. In the subsequent iteration process, we generate the parent generation through binary tournaments, and the individuals in the parent generation generate the offspring through crossover and mutation. Compared with traditional genetic algorithms, we adopt diverse crossover and mutation strategies, our crossover uses single-point crossover and multipoint crossover, and mutation uses sequential mutation and greedy inversion. These diverse operations not only expand the search space, but also help to avoid falling into local optimal solutions. Next, we perform elite selection among the generated offspring and parents, and the result is used as the initial solution for the next iteration. When the fitness of the population does not change or the maximum number of iterations is reached after a certain number of consecutive iterations, the algorithm will terminate and return the current optimal solution.

3) Algorithm Optimization: Due to the fact that genetic algorithm is a population based heuristic algorithm, the large amount of data processed during genetic and selection results in average time efficiency, requiring optimization over time. Firstly, the idea of exchanging space for time requires frequent comparison of whether the distance exceeds the limit in fitness evaluation. Therefore, calculating and saving the distance first can significantly reduce time consumption. In addition, to reduce unnecessary mutation consumption, an adaptive mutation method is adopted to dynamically adjust the mutation rate based on individual fitness values. Individuals with lower fitness can increase their mutation rate to enhance their exploratory ability. Individuals with higher fitness can reduce their mutation rate to maintain their excellent characteristics.

IV. Experiment results

A. Experimental Setup

In the experimental section, we compare the performance of K-Means algorithm, simulated annealing algorithm, genetic algorithm, and EMGA. The constraints include distance between memories and memory power consumption. The number of groups generated by each algorithm and the running time are recorded according to the different constraints. Our experimental platform is:

- CPU: Intel(R) Core(TM) i5-11400H @ 2.70GHz
- Memory: 16GB
- System: Ubuntu 22.04.3 LTS

B. Dataset Description

The specific description of the test sets is shown in Table I. In testcase02 through testcase04, the memory has its own location coordinates for computing the euclidean distance. But testcase01 and testcase05 do not provide location coordinates, so we use the difference in the number of memory hierarchies as a constraint on the distance.

TABLE I: Test Sets Description

Test sets	Number of memories	Power	Position coordinates	From real design
testcase01	69	✓		✓
testcase02	547	✓	✓	✓
testcase03	98	✓	✓	✓
testcase04	25	✓	✓	✓
testcase05	20000	✓		

C. Performance Evaluation

Table II shows the experimental results. Combined number of groups and runtime, EMGA has the best performance.

1) Number of groups: The average number of groups for the four algorithms is shown in Fig. 5. The EMGA shows superior performance in group generation, simulated annealing and genetic algorithms perform slightly worse, and K-Means algorithm has the lowest performance.

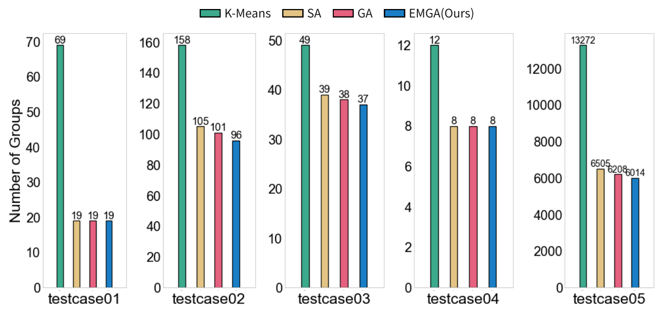


Fig. 5: Average number of groups from testcase01 to testcase05 for different algorithms.

EMGA combines the advantages of greedy and genetic algorithms. It quickly obtains an initial solution and is able to maintain a multi-solution set containing multiple good solutions from the search by an improved genetic algorithm. There may be some differences between these solutions, providing more options. This gives our algorithm better performance than the plain genetic algorithm. In contrast, the simulated annealing algorithm and the K-Means algorithm are more susceptible to the limitations of local search and may fall into local optimal solutions. The simulated annealing algorithm is relatively easy to fall into a certain neighborhood of solutions, leading to the generation of more and similar groupings. The K-Means algorithm itself tends to find local optimal solutions rather than global optimal solutions.

TABLE II: The Number of Groups and Runtime for K-Means, SA, GA, and EMGA

Test sets	Power	Distance(>1e5)/ Hierarchy(<20)	K-Means [10]		SA [11]		GA [9]		EMGA(Ours)		
			Number of groups	Time(s)	Number of groups	Time(s)	Number of groups	Time(s)	Greedy stage	Improved GA stage	Total time(s)
testcase01	0.1	3	68	**0.051**	**17**	0.577	**17**	0.587	17	**17**	0.198
	0.07	2	70	**0.059**	**21**	0.491	**21**	0.580	21	**21**	0.196
	0.11	1	70	**0.061**	**20**	0.551	**20**	0.573	20	**20**	0.198
testcase02	0.1	2000000	87	**0.319**	61	2.785	59	3.002	62	**57**	1.022
	0.06	700000	184	**0.483**	116	2.943	115	2.943	116	**110**	0.823
	0.05	700000	203	**0.584**	139	2.976	129	2.343	130	**122**	0.797
testcase03	0.05	500000	44	**0.020**	34	0.665	32	0.647	35	**31**	0.225
	0.04	500000	50	**0.017**	42	0.698	41	0.654	41	**39**	0.221
	0.04	400000	53	**0.019**	43	0.687	43	0.649	43	**42**	0.222
testcase04	0.04	400000	11	**0.005**	**7**	0.139	**7**	0.281	9	**7**	0.096
	0.03	400000	13	**0.005**	**9**	0.128	**9**	0.280	10	**9**	0.097
	0.03	500000	13	**0.005**	**9**	0.131	**9**	0.303	9	**9**	0.097
testcase05	0.1	5	19687	705.093	14061	91.770	13327	90.128	13259	**13005**	**30.419**
	0.15	8	17794	669.244	4978	141.004	4831	121.713	4713	**4616**	**29.438**
	0.4	11	2335	130.377	478	199.726	466	116.494	435	**423**	**27.688**

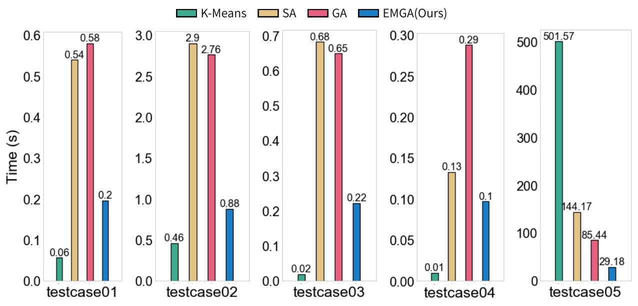


Fig. 6: Average grouping time from testcase01 to testcase05 for different algorithms.

2) Runtime: In Fig. 6, overall EMGA's time is inferior to the K-Menas algorithm, but stronger than the simulated annealing and genetic algorithms. However, in testcase05, the time of our algorithm clearly has the best performance, with the simulated annealing and genetic algorithms having the next best time and K-Means having the longest time.

When using the euclidean distance constraints, because the time complexity of K-Means is relatively low and the distance and power constraints are processed in steps, K-Means has the shortest running time. For test cases without positional coordinates, the gap between memory levels is small, which diminishes the advantage of the K-Means algorithm and greatly increases the runtime of the algorithm. EMGA pre-processes the groupings to reduce the time complexity of the groupings and get a better initial solution. This initial solution is further optimized using an improved genetic algorithm to make the algorithm converge faster and find better results.

Considering both the number of groups and runtime, EMGA performs the best in this experiment. It can generate a smaller number of high-quality groups in a relatively short time. The K-Means algorithm, while advantageous in terms of runtime, sacrifices the quality of the grouping. Due to the use of genetic algorithms, EMGA is inherently parallel as the entire population can evolve at the same time. This allows EMGA to search the solution space somewhat faster, which may be especially important in large-scale problems and high-dimensional spaces.

V. Conclusions

This paper aims to propose a strategy that combines greedy algorithms and genetic algorithms to reduce power consumption, optimize memory block grouping, and improve the overall performance of memory systems. By using greedy algorithms to quickly find local optimal solutions and using them as initial solutions for genetic algorithms, we have successfully improved the convergence speed of the algorithm and the quality of the final solution. This combination strategy fully utilizes the local search advantages of greedy algorithms and the global search characteristics of genetic algorithms. The experimental results show that the proposed algorithm has significant performance improvements in power consumption, distance, and execution time under different test sets and constraints.

VI. Acknowledgment

We are deeply grateful to the reviewers for their valuable comments. Zhou Jin is the corresponding author of this paper. This work was supported by the Natural Science Foundation of China (Grant No. 62204265, 62234010, 62374031, U23A20301).

References

[1] Kang I. Kahng A.B. Co-optimization of memory bist grouping, test scheduling, and logic placement. In *DATE*, 2014.
[2] Rodrigo Z. Reinaldo S., Qadeer Q. Flexible architecture of memory bists. In *LATS*, 2018.
[3] Padmini P. Kriti S.D. Automatic mbist scheduling engine. In *CONECCT*, 2019.
[4] Indranil S. Bibhas G. A distributed bist scheme for noc-based memory cores. In *ECODSD*, 2013.
[5] ShihHsu H. ChanHan Y., ChunHua C. Grouping and placement of memory bist controllers for test application time minimization. In *ISNE*, 2016.
[6] Shoukourian S. Zorian Y. Martirosyan L., Harutyunyan G. A power based memory bist grouping methodology. In *EWDTS*, 2015.
[7] Ratheesh T.V. Darakshan J. Mbist area test time optimization using machine learning. In *ITCDA*, 2023.
[8] Sung-Jin C. Woon-Seek L., Jong-Han H. A heuristic algorithm for a multi-product dynamic lot-sizing and shipping problem. In *IJPE*, 2005.
[9] Jiménez-Fernández S. Carro-Calvo L. Portilla-Figueras J.A. Agustı´n-Blas L.E., Salcedo-Sanz S. A new grouping genetic algorithm for clustering problems. In *ESWA*, 2012.
[10] Shamsul S.M. Mohiuddin A., Raihan S. The k-means algorithm: A comprehensive survey and performance evaluation. In *Electronics*, 2020.
[11] Alsultan K. Shokri Z.S. A simulated annealing algorithm for the clustering problem. In *PTRE*, 1991.