Matrix-based Genetic Algorithm with Principal Component Analysis based Adaptive Parameter Control

Yisu Ge[12], Zhou Zhou[2], Weipeng Tang[2], Jun Zhang[134*]
[1]*College of Computer Science and Technology*, *Zhejiang Normal University*, *Jinhua*, *China*
[2]*College of Computer Science and Artificial Intelligence*, *Wenzhou University*, *Wenzhou*, *China*
[3]*College of Artificial Intelligence*, *Nankai University*, *Tianjing*, *China*
[4]*College of Computing, Hanyang University ERICA, Ansan, South Korea*

***Abstract*—As a classical method for solving complex problems, genetic algorithms (GA) play a significant role in addressing practical issues. However, as the complexity of problems increases, the large number of model variables poses significant challenges for traditional genetic algorithms, including heavy computational burdens and prolonged solving times. Effectively enhancing the problem-solving capabilities of genetic algorithms has become one of the important research topics in the field of artificial intelligence. To this end, this paper focuses on the parameter optimization problem in genetic algorithms utilizing matrix calculations, and proposes a matrix-based genetic algorithm with adaptive parameter control by principal component analysis. To improve its convergence speed in solving large-scale complex problems, the distribution of individuals within the population matrix is compressed and analyzed using principal component analysis, thereby adaptively adjusting the crossover and mutate parameters of genetic algorithm. For applying the matrix-based genetic algorithm on the broader range of problems, the framework is expanded with floating-point encoding. Additionally, to validate the performance of the proposed algorithm, it is compared with the original matrix-based genetic algorithm. The experimental results demonstrate that the new algorithm effectively enhances both the performance and stability of the algorithm.**

***Keywords*—Matrix Genetic Algorithm, Principal Component Analysis, Adaptive Parameter Control**

I. INTRODUCTION

Evolutionary algorithms, inspired by natural processes, are widely used to solve various practical problems, such as the traveling salesman problem[1], multimodal optimization[2-3], electric vehicle charging scheduling[4], supply chain management[5], transportation route optimization[6], and machine scheduling for circuit board drilling[7], et al. As an important branch of artificial intelligence, evolutionary algorithms face enormous challenges with the rapid development of big data technology and computer hardware. High-performance computing devices, such as graphics processing units (GPUs), accelerate the inference and training speeds of models through rapid matrix operations, driving the development of deeper, larger, and more intelligent AI models. (e.g., convolutional and fully connected operations in artificial neural networks) However, there is little research on evolutionary algorithms in this field.

As the scale of the problem continues to expand, the number of variables increases, and the optimization objectives expand, evolutionary algorithms face greater challenges. To address the conflict between problem complexity and limited computational resources in NP hard problems, various optimization algorithms based on evolutionary algorithms are proposed, such as adaptive parameter control, surrogate model-based approximate evaluation, distributed computing, and parallel computing algorithms. Adaptive parameter control algorithms dynamically adjust the hyper parameters of evolutionary algorithms by evaluating the distribution of the population, thereby improving convergence speed. But evaluating the distribution of individuals consumes computational resources, and the resource consumption grows exponentially with the variables number increasing. Surrogate model-based approximate evaluation algorithms reduce computation time by designing or training surrogate models that approximate the original fitness function. However, these models require separate design for different fitness functions, resulting in poor generalization. To further accelerate efficiency, distributed and parallel computing algorithms have emerged. These algorithms utilize multiple computing devices or threads to run the algorithm simultaneously, leveraging more computational resources to improve efficiency. However, data communication and synchronization between multiple threads or devices are the limitation. Furthermore, matrix-based evolutionary algorithms have been proposed to effectively utilize the matrix-computation advantages. These algorithms replace traditional population operators with matrix operations, enabling parallel computation at the individual level. How to design an algorithm only using matrix operation is an interesting challenge. For further improving the matrix-based genetic algorithm, this paper proposes a new framework that fully involves matrix computation to achieve faster convergence.

II. RELATED WORKS

In genetic algorithms, convergence speed is an important metric to evaluating methods. Based on the basic genetic algorithm, adaptive genetic algorithms improve convergence speed by dynamically adjusting parameters. They adjust parameters in real-time according to population characteristics[8-11]. Typically, crossover and mutation

The work is supported in part by National Natural Science Foundation of China (Grant No. 62401398), in part by Zhejiang Provincial Natural Science Foundation of China (Grant No. LQ24F020016), in part by the Wenzhou Key Laboratory Construction Project (Grant No. 2022HZSY0048), in part by the Wenzhou Scientific Research Project (R20250008).

probabilities are the main adjustment content. Yu et al. [9] use a custom discriminant to assess population convergence and adjust crossover and mutation probabilities through macro-control or micro-processing methods to promote convergence. Ding et al. [11] propose an adaptive parameter adjustment method based on individual ranking, improving convergence speed. Magesh et al. [12] propose an Adaptive Genetic Algorithm integrated with Reinforcement Learning (AGA-RL), which modifies the GA's key parameters such as crossover and mutation probabilities based on real-time environmental feedback to adapt to the rapidly changing channel conditions in quantum and nano communication systems, thereby accelerating the algorithm's convergence speed. The adaptive genetic algorithms are the useful way to accelerate convergence only modifying hyper parameters. With the expanding of variables number, explore a simple way to evaluate the population distribution is the main point in the future work.

Li et al. [13] mathematically analyze the total cost of expensive optimization problems and summarize three directions for reducing costs: approximate evaluation, algorithm enhancement, and parallel and distributed computing. Surrogate model-based approximate evaluation algorithms reduce computation costs during the evaluation phase by designing or training surrogate models. Jin et al. [14] introduce various surrogate models, including2 polynomials, Gaussian processes, and neural networks. Luo et al. [15] investigate data-driven dynamic optimization and develop a surrogate-assisted framework for solving such problems. Although surrogate models reduce evaluation costs during algorithm execution, they require separate design or training for each problem, incurring additional upfront costs.

Originally, evolutionary computation algorithms ran on single-core CPUs, with all steps executed serially. As problem complexity increased, parallel or distributed evolutionary algorithms [16]-[20]became a focus of research. The basic idea is to distribute the computational load across more computing units, with appropriate information exchange methods designed to ensure optimization efficiency. Another approach is to parallelize computations within a single population iteration.

For full using the technology development to accelerate the evaluation algorithm, Zhan et al. [21] propose the Matrix-based Genetic Algorithm (MGA) framework, which parallelizes the computational flow of traditional genetic algorithms using matrix operations. This matrix-based approach makes the algorithm easier to fit into different high-performance computing hardware, which benefits from a rich software ecosystem for matrix computations. Although matrix-based evolutionary algorithm slightly increases computational complexity, it effectively utilizes parallel computing resources while avoiding the shortcomings of data transmission and synchronization in population-level parallel methods, achieving truly efficient parallel evolutionary computation. Building on the matrix-based parallelization idea of the MGA framework, Akopov et al. [22] proposes a Matrix-Based Hybrid Genetic Algorithm (MBHGA) tailored for agent-based models of firms' behavior with controlled trade interactions. Unlike the original MGA that focuses on optimizing computational flow via matrix operations, MBHGA introduces a hybrid evolutionary mechanism by integrating real-coded crossover and matrix binary-coded crossover as core genetic operators, addressing the specific demand for strategy optimization in multiagent systems.

To further improve the efficiency of MGA, this paper integrates adaptive parameter adjustment methods based on population distribution analysis. By using matrix-based principal component analysis (PCA) to deconstruct the distribution of individuals in the population matrix, the algorithm adaptively adjusts crossover and mutation parameters based on the convergence degree of the population matrix. The main contributions of this paper are as follows:

- Estimating the distribution of the population matrix by matrix-based PCA and adaptively adjusting crossover and mutation parameters to dynamically guide the population toward rapid convergence.
- Extending the existing matrix-based evolutionary algorithm framework with floating-point encoding, including matrix-based simulated binary crossover (MSBX) and matrix-based polynomial mutation (MPM), to facilitate its application to continuous problems.
- Comparing the proposed PCA-based adaptive parameter control MGA with the original matrix-based genetic algorithm to validate the effectiveness of the proposed method.

The remainder of this paper is organized as follows: Section1 is the introduction, and the related works are presented in Section 2. Section 3 details the matrix-based genetic algorithm and its extension with floating-point operators, including the basic operations and computational expressions of MGA and the matrix-based improvements of genetic operators used in this paper. Section 4 introduces the PCA-based adaptive parameter control and the algorithm's workflow. The experiment are introduced in Section 5, and Section 6 concludes the paper.

III. Matrix-based Algorithm

In recent years, the matrix-based approach to evolutionary algorithms has emerged as a cutting-edge research direction. The algorithm proposed in this paper is based on the matrix-based genetic algorithm (MGA). This section introduces the foundational concepts of the matrix-based genetic algorithm. From a logical perspective, the matrix-based genetic algorithm is almost indistinguishable from traditional genetic algorithms. However, in terms of implementation, operations such as selection, crossover, and mutation are performed using matrix computations.

A. Matrix and Parameter Descriptions

For ease of reading, Table. 1 provides the matrices, operations and their descriptions used in this paper. The subscripts of the matrices indicate their dimensions.

B. Algorithm Basics

The proposed algorithm is based on the MGA framework introduced in [21]. For parts such as initialization, evaluation, and selection that remain largely unchanged, a brief description is provided.

TABLE I. MATRICES, OPERATIONS, AND DESCRIPTIONS

Notations	Descriptions
X	Population matrix, representing all individuals in the population.
Fit	Fitness matrix, representing the fitness values of all individuals in the population.
Z_N	Anti-diagonal identity matrix, as: $Z_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$ used to reverse the order of row vectors in other matrices.
ONE	All-ones matrix, where all elements are 1.
R	Random matrix, where elements are randomly distributed in [0, 1].
U	Upper bound matrix, storing the upper bounds of parameters.
L	Lower bound matrix, storing the lower bounds of parameters.
$Prob$	Selection probability matrix, storing the probabilities of individuals being selected.
$LTRI$	Lower triangular identity matrix, where lower triangular elements are 1 and others are 0.
SEL	Selection matrix, storing the indices of individuals in the new population from the old population.
B	Matrix storing the β from the SBX in MSBX.
G	Matrix storing the g from the original PM in MPM.
J	Matrix storing the j values from the original PM in MPM.
D	Matrix storing the perturbation coefficients from the original PM in MPM.
M	Matrix composed of 0s and 1s, indicating whether the corresponding elements undergo mutation.
H	Matrix storing the maximum change values of variables from the original PM in MPM.
UT	A square matrix as: $UT = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ with $1's$ on the diagonal and upper triangular elements, and $0's$ elsewhere.
MQ	The matrix formed by removing the last column from an identity matrix, as: $MQ = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$。
MR	The matrix obtained by removing the first column of an identity matrix, as: $MR = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$。
$\times$	Matrix multiplication.
$\circ$	Hadamard product (element-wise multiplication).
Div	Element-wise division of two matrices, as: $C = Div(A, B)$, where $c = a/b$ for corresponding elements a, b, c in A, B, C.
$<$	Element-wise logical less-than operation, returning 1 if true and 0 otherwise (similarly for >).
$ExInd$	Indexing operator.

In the matrix-based genetic algorithm, to perform iterations in matrix form, an individual is represented as a row vector with D elements, as follows:

$$S_i = (s_{i1}, s_{i2}, \dots, s_{iD}) \tag{1}$$

where i is the index of the individual, which also corresponds to the row number in the matrix, $1 \le i \le N$. The entire population is represented as an $N \times D$ matrix X, as follows:

$$X_{N\times D} = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_N \end{pmatrix} \tag{2}$$

Initialization is performed as follows:

$$X_{N\times D} = ONE_{N\times 1} \times (U - L)_{1\times D} \circ R_{N\times D} + ONE_{N\times D} \times L_{1\times D} \tag{3}$$

The selection operation is applied to the entire population matrix $X_{N\times D}$. The fitness computation operation $CptF(\cdot)$ is applied to obtain the fitness vector $Fit_{N\times 1}$:

$$F_{N\times 1} = CptF(X_{N\times D}) \tag{4}$$

$F_{N\times 1}$ is left-multiplied by $LTRI_{N\times N}$ and normalized to obtain $Prob_{N\times 1}$:

$$Prob_{N\times 1} = Nor(LTRI_{N\times N} \times F_{N\times 1}) \tag{5}$$

$Nor(\cdot)$is normalization opteration. $Prob_{N\times 1}$ is transposed and expanded into $Prob_{N\times N}$:

$$Prob_{N\times N} = ONE_{N\times 1} \times Trans(Prob_{N\times 1}) \tag{6}$$

A matrix $CR_{N\times N}$ is generated, where each row contains identical random numbers between 0 and 1. The logical operation $Prob_{N\times N} < CR_{N\times N}$ is performed, and the result is right-multiplied by $ONE_{N\times 1}$ to obtain the selection index vector $SEL_{N\times 1}$:

$$SEL_{N\times 1} = (Prob_{N\times N} < CR_{N\times N}) \times ONE_{N\times 1} \tag{8}$$

$$CR_{N\times N} = R_{N\times 1} \times ONE_{1\times N} \tag{9}$$

The selection operation is completed by applying the indexing operator $ExInd(\cdot)$ to the pre-selection population matrix using $SEL_{N\times 1}$:

$$X_{N\times D} = ExInd(X_{N\times D}, SEL_{N\times 1}) \tag{10}$$

Simulated Binary Crossover (SBX) and Polynomial Mutation (PM) are common crossover and mutation operators for real-coded genetic algorithms. This paper matrixizes these operators, as described below.

C. Matrix-Based Simulated Binary Crossover (MSBX)

Given two parent individuals x^1 and x^2, the SBX operator generates two offspring individuals c^1 and c^2 as follows:

$$\begin{cases} c^1 = 0.5\big((1+\beta)x^1 + (1-\beta)x^2\big) \\ c^2 = 0.5\big((1-\beta)x^1 + (1+\beta)x^2\big) \end{cases} \tag{11}$$

where β is calculated as:

$$\beta = \begin{cases} (2r)^{\frac{1}{1+g}}, r \le 0.5 \\ \dfrac{1}{(2-2r)^{\frac{1}{1+g}}}, otherwise \end{cases} \tag{12}$$

where r is a random number uniformly distributed in [0, 1], and g is a user-defined parameter. A larger g increases the probability of offspring being closer to the parents.

To facilitate element-wise matrix operations, the operation HP is defined, which performs exponentiation on corresponding elements of two matrices:

$$C = HP(A, B) \tag{13}$$

where A, B, C are matrices of the same size. For corresponding elements $a_{i,j}, b_{i,j}, c_{i,j}$ in A, B, C:

$$c_{i,j} = {a_{i,j}}^{b_{i,j}} \tag{14}$$

In the MGA framework, parent individuals for crossover are selected based on the crossover probability pc. The number of individuals participating in crossover N_c is:

$$N_c = N \times pc \tag{15}$$

The portion of $X_{N\times D}$ participating in crossover $X_{N_c\times D}$ is defined as:

$$X_{N_c\times D} = \{X_{N\times D}|1,2,\dots N_c\} \tag{16}$$

Considering that the original form of SBX requires selecting pairs of parents for crossover, this paper proposes a method to enable the operator to be performed in a matrix-based manner. The proposed approach involves reversing the order of all row vectors in the population matrix block $X_{N_c\times D}$ participating in crossover, resulting in a new matrix block denoted as $DX_{N_c\times D}$:

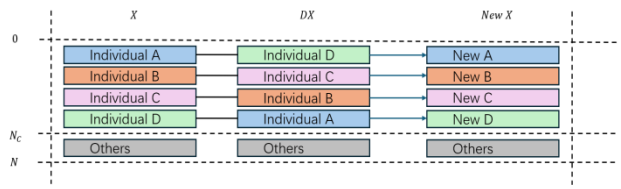

Fig. 1. Schematic Diagram of Parent Individual Pairing.

$$DX_{N_c\times D} = Z_{N_c} \times X_{N_c\times D} \tag{17}$$

Pairs of individuals from the same rows of $X_{N_c\times D}$ and $DX_{N_c\times D}$ are selected as parents for crossover, as illustrated in Fig. 1. The offspring population $C_{N_c\times D}$ is obtained as follows:

$$C_{N_c\times D} = 0.5 \cdot \begin{pmatrix} (ONE_{N_c\times D} + B_{N_c\times D}) \circ X_{N_c\times D} + \\ (ONE_{N_c\times D} - B_{N_c\times D}) \circ DX_{N_c\times D} \end{pmatrix} \tag{18}$$

Referring to SBX, the value of β in Equations (11) and (12) is determined by a conditional function. Each pair of offspring requires an independent random value r for branch judgment. In MSBX, these random values exist in the form of a matrix $R_{N_c\times 1}$. To enable this process to be performed in a matrix-based manner, this paper replaces the dual-branch conditional judgment with logical operations. The β value matrix $B_{N_c\times D}$ in MSBX is obtained as follows:

$$B_{N_c\times D} = T^1_{N_c\times 1} \times ONE_{1\times D} \circ Hst + T^2_{N_c\times 1} \times ONE_{1\times D} \circ Hnd \tag{19}$$

$$T^1_{N_c\times 1} = R_{N_c\times 1} \le \left(\frac{ONE_{N_c\times 1}}{2}\right) \tag{20}$$

$$T^2_{N_c\times 1} = ONE_{N_c\times 1} - T^1_{N_c\times 1} \tag{21}$$

$$Hst = HP\begin{pmatrix} 2\cdot R_{N_c\times D}, Div(ONE_{N_c\times D}, ONE_{N_c\times D} \\ +G_{N_c\times D}) \end{pmatrix} \tag{22}$$

$$Hnd = HP\begin{pmatrix} Div\left(ONE_{N_c\times D}, 2\cdot\left(ONE_{N_c\times D} - R_{N_c\times D}\right)\right), \\ Div(ONE_{N_c\times D}, ONE_{N_c\times D} + G_{N_c\times D}) \end{pmatrix} \tag{23}$$

Algorithm1: Matrix-Based Simulated Binary Crossover (*MSBX*)

1:Input: Population X

2:Generate R

3:$T1 = R < (ONE/2)$

4:$T2 = ONE - T1$

5:Calculate Hst, Hnd with (22), (23)

6:$B = T1 \times ONE \circ Hst + T2 \times ONE \circ Hnd$

7:C= $((ONE + B) \circ X + (ONE - B) \circ DX)/2$

8:Output: C

End

D. Matrix-Based Polynomial Mutation (MPM)

Assume that a gene $gene$ in a parent individual x undergoes mutation. Using the PM operator, the offspring gene $cgene$ is generated as follows:

$$cgene = gene + d \times l \tag{24}$$

where l represents the maximum change value of the element, which is related to the variable's range. The perturbation coefficient d is calculated as:

$$d = \begin{cases} (2r)^{\frac{1}{1+j}} - 1, r \le 0.5 \\ 1 - (2-2r)^{\frac{1}{1+j}}, otherwise \end{cases} \tag{25}$$

where, j is a parameter that influences the distribution of the gene after mutation. Below is a brief explanation of the MPM:

The pre-mutation matrix is $X_{N\times D}$, and the offspring matrix $C_{N\times D}$ generated by polynomial mutation is obtained as:

$$C_{N\times D} = X_{N\times D} + M_{N\times D} \circ D_{N\times D} \circ H_{N\times D} \tag{26}$$

where $D_{N\times D}$ is calculated as:

$$D_{N\times D} = T^3_{N\times D} \circ (Hrd - ONE_{N\times D}) + T^4_{N\times D} \circ (ONE_{N\times D} - Hth) \tag{27}$$

$$T^3_{N\times D} = R_{N\times D} \leq \left(\frac{ONE_{N\times D}}{2}\right) \tag{28}$$

$$T^4_{N\times D} = ONE_{N\times D} - T^3_{N\times D} \tag{29}$$

$$Hrd = HP(2R_{N\times D}, Div(ONE_{N\times D}, ONE_{N\times D} + J_{N\times D})) \tag{30}$$

$$Hth = HP\begin{pmatrix} 2ONE_{N\times D} - 2R_{N\times D}, \\ Div(ONE_{N\times D}, ONE_{N\times D} + J_{N\times D}) \end{pmatrix} \tag{31}$$

Algorithm2: Matrix-Based Polynomial Mutation (MPM)

1:Input: Population X

2:Generate $R1, R2$

3:$H = U - L$

4:$M = R2 < POM$

5:$T3 = R2 < (ONE/2)$

6:$T4 = ONE - T1$

7:Calculate Hrd, Hth with (30), (31)

8:$D = T3 \circ (Hrd - ONE) + T4 \circ (ONE - Hth)$

9:$C = X + M \circ D \circ H$

10:Output: C

End

IV. PCA-BASED PARAMETER ADJUSTMENT STRATEGY

To enhance the performance of the algorithm, the parameters can be adjusted based on the state of the population. This paper employs Principal Component Analysis (PCA) to analyze the population's state. PCA is an effective dimensionality reduction method that extracts the principal features of data. By using PCA, we can reduce the dimensionality of the data, simplify its complexity, and extract key features for further analysis and processing. The steps for performing PCA on matrix X are as follows:

Step 1: calculate the standard deviation for each row to obtain the standard deviation vector Std of the matrix. Then compute the normalized matrix X_{std} using Equation (32).

$$X_{std} = Div\big((X - \bar{X}), Std\big) \tag{32}$$

where $\bar{X}$ is the matrix with elements being the row-wise means.

Step 2: Compute the covariance matrix Q using Equation (33).

$$Q = \frac{X^T_{std} \times X_{std}}{n} \tag{33}$$

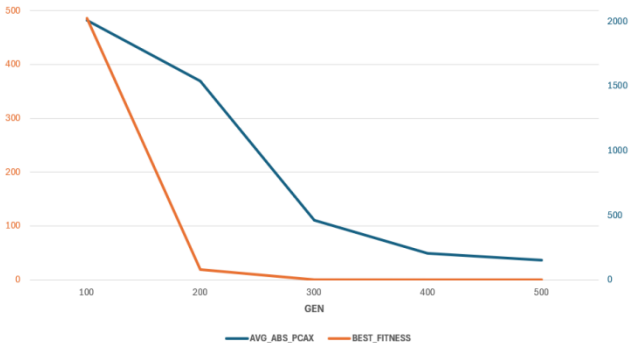

Fig. 2. Variation of *AAP* and FIT Over Iterations.

where X^T_{std} denotes the transpose matrix of X_{std}.

Step 3: Compute the eigenvalues and arrange the corresponding eigenvectors into matrix V, as shown in Equation (34).

$$V = Eig(Q) \tag{34}$$

where $Eig(\cdot)$ denotes the operator that computes and sorts the eigenvectors of the original data matrix.

Step 4: Perform full projection on the original data X to obtain the result $PCAX$, as shown in Equation (35).

$$PCAX = X \times V \tag{35}$$

For an existing population $X_{N\times D}$, the data is compressed using $PCA(\cdot)$. Based on the top K eigenvectors with the largest eigenvalues, the compressed data $PCAX_{K\times D}$ is obtained:

$$PCAX_{K\times D} = PCA(X_{N\times D}, K) \tag{36}$$

where K is the target number of rows for the compressed data, $1 \leq K \leq N$. In this paper, $K = 1$ is chosen, and the metric AAP for evaluating the original population is calculated as:

$$AAP = AVG(ABS(PCAX_{1\times D}) \tag{37}$$

Here, $ABS(\cdot)$ denotes taking the absolute value of the elements in the matrix, and $AVG(\cdot)$ represents the average of all elements in the matrix.

To explore the relationship between the PCA-compressed data and the population distribution state, as well as to investigate how PCA-processed data can be utilized for parameter control, this paper conducted preliminary experiments. The experiments were performed using the MGA framework over 500 iterations, with PCA compression applied to the population every 100 iterations. During this process, the best fitness value (FIT) and the AAP metric were recorded. Fig. 2 illustrates the variation of AAP and the FIT throughout the iteration process in one of the experiments. As a preliminary exploratory experiment, individuals were represented using floating-point encoding. However, the operators were not optimized for floating-point encoding but instead utilized the basic operators from the MGA framework.

It is observed a that when AAP is at a higher level, the population converges at a faster rate. Conversely, when AAP decreases, the convergence speed of the population also slows down. This suggests that AAP may be correlated with the

quality of the population's distribution. The magnitude of $PCAX_{1\times D}$ reflects the degree of population concentration. As the MGA iterations progress, the magnitude of $PCAX_{1\times D}$ tends to decrease. Because PCA is a data compression algorithm, the more similar the compressed data, the smaller the magnitude of $PCAX_{1\times D}$. This observation aligns with the intuitive understanding that the population gradually converges during the genetic algorithm process. The AAP metric provides a novel perspective for interpreting the population's distribution state.Considering the relatively high computational cost of PCA, this paper proposes the following strategy:

Algorithm3: PCA-based Adaptive Parameter Adjustment Strategy

1: If GEN % $ST == 0$
2: If $GEN == 0$
3: $OAAP == Current\ AAP$
4: $OFIT == Best\ FIT$
5: Else
6: $NAAP == Current\ AAP$
7: $NFIT == Best\ FIT$
8: Refresh P_c, P_m with (38-43)
9: Refresh a, f with (44), (45)

End

During the population iteration process, PCA is applied to the population matrix every ST iterations to record the population's state, including AAP and best fitness (FIT), and to perform an adaptive parameter update. The newly recorded AAP and FIT are denoted as $NAAP$ and $NFIT$, respectively, while the previously recorded AAP and FIT are denoted as $OAAP$ and $OFIT$. The adjustment of crossover probability P_c and mutation probability P_m is based on the following four dynamic indicators:

$$dP_{c,1} = 2\left(\frac{NFIT}{OFIT} > f\right) - 1 \tag{38}$$

$$dP_{c,2} = 2\left(\frac{NAAP}{OAAP} > a\right) - 1 \tag{39}$$

$$dP_{m,1} = 2\left(\frac{NFIT}{OFIT} < f\right) - 1 \tag{40}$$

$$dP_{m,2} = 2\left(\frac{NAAP}{OAAP} < a\right) - 1 \tag{41}$$

The rules for updating P_c and P_m are as follows:

$$P_c = P_c + k_1(dP_{c,1} + dP_{c,2}) \tag{42}$$

$$P_m = P_m + k_2(dP_{m,1} + dP_{m,2}) \tag{43}$$

where k_1 and k_2 are the base adjustment factors for P_c and P_m respectively. After updating P_c and P_m, the values of f and a are updated as follows:

$$f = f - k_3(dP_{m,1} - dP_{c,1}) \tag{44}$$

$$a = a - k_4(dP_{m,2} - dP_{c,2}) \tag{45}$$

where k_3 and k_4 are the base adjustment factors for f and a respectively.

V. EXPERIMENTS

To validate the performance and efficiency of the proposed method, this paper compares the original MGA with the PCA-enhanced MGA (MGA_P) on a set of test functions. The test functions are as Table. 2.

TABLE II. MATRIX-BASED TEST FUNCTIONS

Function	Range
$F1_{N\times1} = (X_{N\times D} \circ X_{N\times D}) \times ONE_{D\times1}$	[-5.12,5.12]
$F2_{N\times1} = (X_{N\times D} \times UT_{D\times D}) \circ (X_{N\times D} \times UT_{D\times D}) \times ONE_{D\times1}$	[-5.12, 5.12]
$F3_{N\times1} = MAX\_R(ABS(X_{N\times D}))$	[-10, 10]
$F4_{N\times1} = INT\_B(X_{N\times D} + 0.5 \circ ONE_{N\times D}) \circ INT\_B(X_{N\times D} + 0.5 \circ ONE_{N\times D}) \times ONE_{D\times1}$	[-5.12, 5.12]
$F5_{N\times1} = (X_{N\times D} \circ X_{N\times D} \circ X_{N\times D}) \times MG_{N\times1} + R_{N\times1}$	[-1.28, 1.28]
$F6_{N\times1} = (100 \circ (HP(X_{N\times D} \times MR_{D\times(D-1)} - HP(X_{N\times D} \times MQ_{D\times(D-1)}, 2), 2) + HP(X_{N\times D} \times MQ_{D\times(D-1)} - ONE_{N\times(D-1)}, 2))) \circ ONE_{(D-1)\times1}$	[-10, 10]
$F7_{N\times1} = -X_{N\times D} \circ SIN\left(SQRT(ABS(X_{N\times D}))\right) \times ONE_{D\times1} + 418.98 \circ D \circ ONE_{N\times1}$	[-500, 500]
$F8_{N\times1} = (X_{N\times D} \circ X_{N\times D} - 100 \circ COS(2\pi \circ X_{N\times D}) + 10 \circ ONE_{N\times D}) \times ONE_{D\times1}$	[-5.12, 5.12]

TABLE III. EXPERIMENTAL PARAMETER SETTINGS

Variable	Value	Descriptions
N	20,50,100	Population Size
D	5	Variable Dimension
$MAXGEN$	100,200,500	Maximum Iteration Count
$Initiated\ Pc$	0.7	Initial Crossover Probability
$Initiated\ Pm$	0.05	Initial Mutation Probability
ST	20	Parameter Update Interval
g	5	Parameter g in Equation (12)
j	0.2	Parameter j in Equation (25)
f	0.8	Parameter f in Equation (38)
a	0.5	Parameter a in Equation (39)
k_1	0.05	Parameter k_1 in Equation (42)
k_2	0.005	Parameter k_2 in Equation (43)
k_3	0.08	Parameter k_3 in Equation (44)
k_4	0.08	Parameter k_4 in Equation (45)
$RERUN$	30	Repeated Experiments.

A. Implementation Details

The experiments were conducted in a C++ environment, with matrix-based computations implemented using the Eigen library. The hardware configuration includes an Intel Core i5-12600k processor with 48 GB of RAM. The baseline parameter settings for the algorithm and their descriptions are shown in Table. 3.

The experiments were divided into three groups based on the maximum number of iterations (100, 200, and 500). For each group, tests were conducted with population sizes of 50, 100, and 200. Each configuration was repeated 30 times, and the average and standard deviation of the best fitness values were recorded.

B. Experimental Results

The following Table.4-Table. 6 present the performance and stability of the improved algorithm (MGA_P) and the original algorithm (MGA) under different maximum iterations and

TABLE IV. COMPARATIVE EXPERIMENTS UNDER MAXGEN = 100

MGA_P	N=20		N=50		N=100	
	μ	σ	μ	σ	μ	σ
f_1	**8.07E-02**	**3.26E-01**	**1.98E-02**	**1.02E-01**	**5.85E-03**	**2.99E-02**
f_2	2.05E+00	9.88E+00	**7.89E-01**	4.56E+00	**3.00E-01**	**1.73E+00**
f_3	**1.67E+00**	**5.01E+00**	**4.46E-01**	1.14E+00	**1.95E-01**	**6.36E-01**
f_4	3.66E-01	**2.63E+00**	**2.33E-01**	**2.31E+00**	**1.33E-01**	**1.86E+00**
f_5	**1.35E-01**	**4.44E-01**	**7.12E-02**	2.46E-01	**3.46E-02**	**1.53E-01**
f_6	**5.42E+01**	2.96E+02	**2.03E+01**	**7.82E+01**	5.57E+00	3.23E+01
f_7	**2.85E+02**	**7.40E+02**	2.70E+02	5.74E+02	**1.40E+02**	**3.74E+02**
f_8	**4.12E+00**	**1.31E+01**	**2.10E+00**	**8.90E+00**	**9.47E-01**	**5.43E+00**
MGA	**N=20**		**N=50**		**N=100**	
	μ	σ	μ	σ	μ	σ
f_1	1.29E-01	7.34E-01	2.60E-02	1.35E-01	7.38E-03	4.59E-02
f_2	**1.99E+00**	**9.38E+00**	9.97E-01	**4.23E+00**	5.25E-01	3.93E+00
f_3	1.71E+00	5.65E+00	4.65E-01	**1.03E+00**	2.08E-01	6.77E-01
f_4	**3.33E-01**	4.32E+00	2.66E-01	2.42E+00	**1.33E-01**	**1.86E+00**
f_5	1.57E-01	7.61E-01	7.88E-02	**2.31E-01**	3.75E-02	2.06E-01
f_6	5.69E+01	**2.28E+02**	2.14E+01	8.31E+01	**4.37E+00**	**2.26E+01**
f_7	3.53E+02	8.46E+02	**2.22E+02**	**5.15E+02**	1.59E+02	5.35E+02
f_8	5.58E+00	1.90E+01	2.45E+00	9.03E+00	2.17E+00	6.66E+00
	6,2	6,2	7,1	4,4	6,2	6,2

TABLE V. COMPARATIVE EXPERIMENTS UNDER MAXGEN = 200

MGA_P	N=20		N=50		N=100	
	μ	σ	μ	σ	μ	σ
f_1	**1.76E-02**	**8.21E-02**	**4.83E-03**	1.85E-02	**9.93E-04**	6.62E-03
f_2	1.28E+00	6.03E+00	6.22E-01	**1.81E+00**	1.96E-01	4.59E-01
f_3	4.96E-01	5.01E+00	**2.46E-01**	6.42E-01	**1.03E-01**	**3.16E-01**
f_4	**1.00E-01**	**1.64E+00**	1.00E-01	1.64E+00	6.66E-02	1.36E+00
f_5	**9.36E-02**	**3.56E-01**	**4.02E-02**	**1.29E-01**	**2.19E-02**	8.59E-02
f_6	**1.66E+01**	**1.48E+02**	**9.36E+00**	**2.35E+01**	**3.57E+00**	1.14E+01
f_7	**2.92E+02**	**3.67E+02**	2.43E+02	4.80E+02	**2.00E+02**	**3.92E+02**
f_8	**2.24E+00**	**7.65E+00**	**8.82E-01**	**5.80E+00**	**2.61E-01**	**2.92E+00**
MGA	**N=20**		**N=50**		**N=100**	
	μ	σ	μ	σ	μ	σ
f_1	2.13E-02	1.02E-01	5.53E-03	**1.26E-02**	1.41E-03	**5.00E-03**
f_2	**1.19E+00**	**5.25E+00**	**5.96E-01**	3.16E+00	**8.87E-02**	**4.12E-01**
f_3	**4.88E-01**	**3.96E+00**	2.71E-01	**5.92E-01**	1.17E-01	3.45E-01
f_4	1.33E+00	1.86E+00	**6.66E-02**	**1.36E+00**	**3.33E-02**	**9.83E-01**
f_5	1.13E-01	4.14E-01	4.63E-02	2.24E-01	2.23E-02	**8.05E-02**
f_6	1.79E+01	1.52E+02	1.08E+01	2.39E+01	3.65E+00	**1.08E+01**
f_7	2.97E+02	3.81E+02	**2.08E+02**	**3.96E+02**	2.35E+02	4.73E+02
f_8	3.37E+00	1.02E+01	1.12E+00	6.43E+00	5.41E-01	5.28E+00
	6,2	6,2	5,3	4,4	6,2	3,5

TABLE VI. COMPARATIVE EXPERIMENTS UNDER MAXGEN = 500

MGA_P	N=20		N=50		N=100	
	μ	σ	μ	σ	μ	σ
f_1	**6.34E-03**	**3.56E-02**	**1.23E-03**	**2.93E-03**	1.96E-04	7.34E-04
f_2	**8.13E-01**	**4.33E+00**	**2.39E-01**	**1.09E+00**	**1.55E-01**	6.59E-01
f_3	2.40E-01	**1.21E+00**	**1.02E-01**	2.34E-01	**8.95E-02**	7.36E-02
f_4	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	1.00E-01	1.64E+00
f_5	7.22E-02	3.36E-01	**1.86E-02**	**6.06E-02**	**1.47E-02**	**5.54E-02**
f_6	**3.29E+00**	**1.78E+01**	2.15E+00	1.53E+01	**6.14E-01**	1.82E+00
f_7	**2.31E+02**	5.11E+02	**2.07E+02**	**3.47E+02**	**1.34E+02**	**3.65E+02**
f_8	**9.35E-01**	**4.47E+00**	**1.64E-01**	**1.07E+00**	**3.97E-02**	2.49E-01
MGA	**N=20**		**N=50**		**N=100**	
	μ	σ	μ	σ	μ	σ
f_1	1.56E-02	7.12E-02	2.26E-03	5.93E-03	**1.74E-04**	**6.58E-04**
f_2	1.20E+00	6.82E+00	3.77E-01	1.44E+00	1.80E-01	**5.32E-01**
f_3	**1.67E-01**	2.01E+00	1.15E-01	**2.15E+00**	8.97E-02	**6.76E-01**
f_4	1.33E-01	1.86E+00	1.00E-01	1.64E+00	**0.00E+00**	**0.00E+00**
f_5	**5.54E-02**	**1.86E-01**	2.19E-02	7.34E-02	1.56E-02	6.04E-02
f_6	4.01E+00	2.32E+01	**2.03E+00**	**1.42E+01**	6.55E-01	**1.79E+00**
f_7	2.89E+02	**4.89E+02**	2.11E+02	3.93E+02	1.57E+02	4.14E+02
f_8	1.78E+00	6.29E+00	5.97E-01	3.14E+00	5.35E-02	**2.00E-01**
	6,2	6,2	7,1	6,2	6,2	2,6

population sizes. The last row of each table indicates the number of times each algorithm outperformed the other across the test functions under the same maximum iterations and population size.

The above tests demonstrate that, under various test conditions, the performance and stability of MGA improved after incorporating the PCA-based parameter adjustment strategy. This indicates that the PCA-based adaptive parameter control strategy has a positive effect. Compared to modifying

crossover and mutation operators, adaptive parameter control strategies typically provide limited improvements to the convergence performance of algorithms, especially when the initial parameters are already near optimal. However, as an additional dimension to aid algorithm convergence, adaptive parameter control holds significant value for high-performance algorithms.

VI. Conclution

To further enhance the performance and efficiency of matrix-based evolutionary algorithms, this paper improves upon the original matrix-based genetic algorithm (MGA) in two main aspects. First, by leveraging Principal Component Analysis (PCA), the distribution of the population is analyzed in a matrix-based manner, and the parameters of the evolutionary algorithm are adaptively controlled to improve performance. Second, to address scenarios where floating-point encoding is more suitable, this paper extends the original matrix-based framework by incorporating matrix-based implementations of commonly used floating-point operators, such as Simulated Binary Crossover (SBX) and Polynomial Mutation (PM), making the algorithm more versatile for a wider range of problems. Experimental results demonstrate that the proposed method improves the convergence performance of the algorithm. However, in cases with larger population sizes, the overall runtime increases due to the higher time complexity of PCA.

Given the relatively high computational cost of PCA in the test functions, this paper limits the frequency of PCA usage. This constraint results in a relatively coarse-grained parameter control strategy based on PCA results. Therefore, the proposed algorithm has potential for further improvement in adaptive parameter control, requiring more experiments and discussions to validate. Future work will focus on addressing the high computational cost of PCA and exploring better parameter control methods. These improvements will further enhance the performance and applicability of the proposed algorithm.

References

[1] M. A. Alhanjouri and B. Alfarra, "Ant colony versus genetic algorithm based on travelling salesman problem," Int. J. Comput. Tech. Appl, vol. 2, no. 3, pp. 570-578, 2013.

[2] T. Huang, Y.-J. Gong, W.-N. Chen, H. Wang, and J. Zhang, "A probabilistic niching evolutionary computation framework based on binary space partitioning," IEEE transactions on cybernetics, 2020.

[3] Z.-G. Chen, Z.-H. Zhan, H. Wang, and J. Zhang, "Distributed individuals for multiple peaks: A novel differential evolution for multimodal optimization problems," IEEE Transactions on Evolutionary Computation, vol. 24, no. 4, pp. 708-719, 2019.

[4] W.-L. Liu, Y.-J. Gong, W.-N. Chen, Z. Liu, H. Wang, and J. Zhang, "Coordinated charging scheduling of electric vehicles: A mixed-variable differential evolution approach," IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 12, pp. 5094-5109, 2019.

[5] T. S. Rao, "An Evaluation of ACO and GA TSP in a Supply Chain Network," Materials Today: Proceedings, vol. 5, no. 11, pp. 25350-25357, 2018.

[6] U. Hacizade and I. Kaya, "Ga based traveling salesman problem solution and its application to transport routes optimization," IFAC-PapersOnLine, vol. 51, no. 30, pp. 620-625, 2018.

[7] Z.-W. Zhou and T.-M. Ding, "Research on holes machining path planning optimization with TSP and GA," Modular Machine Tool & Automatic Manufacturing Technique, vol. 66, no. 7, pp. 30-32, 2007.

[8] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, no. 4, pp. 656-667, April 1994.

[9] G.-S. Yu. and X.-W. Yu, "An improved adaptive genetic algorithm," Mathematics in Practice and Theory, vol. 45, no.19, pp. 259-264, 2015.

[10] C.-R. Yang, Q. Qian and F. Wang, "Application of improved adaptive genetic algorithm in function optimization,"Journal of Computer Applications, vol. 35, no. 4, pp. 1042-1045, 2018.

[11] J-. H. Ding and Z.-J. Zhang,"Adaptive Genetic Algorithm Based on Individual Ordering," Electronic Science and Technology, vol. 33, no. 3, pp. 6-11, 2020.

[12] Magesh G. Quantum Channel Optimization: Integrating Quantum-Inspired Machine Learning With Genetic Adaptive Strategies[J]. IEEE Access, 2024, 12: 80397-80417.

[13] J.-Y. Li, Z.-H. Zhan and J. Zhang, "Evolutionary Computation for Expensive Optimization: A Survey," Mach. Intell. Res. 19, pp. 3–23, 2022.

[14] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation,"Soft Computing, vol. 9, no. 1, pp. 3–12, 2005.

[15] W. Luo, R. Yi, B. Yang and P. Xu, "Surrogate-Assisted Evolutionary Framework for Data-Driven Dynamic Optimization," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 3, no. 2, pp. 137-150, 2019.

[16] M. Amirghasemi, "An effective parallel evolutionary metaheuristic with its application to three optimization problems, " Appl. Intell. , vol. 53, pp. 5887–5909, 2023.

[17] M. Sevkli and M. E. Aydin, "Parallel variable neighbourhood search algorithms for job shop scheduling problems," IMA. J. Manag. Math. , vol. 18, no. 2, pp. 117–133, 2007.

[18] S. Stöppler and C. Bierwirth, "The application of a parallel genetic algorithm to the max flowshop problem," A New directions for operations research in manufacturing, section 10, 1992.

[19] Z.-J. Wang, Z.-H. Zhan, and J. Zhang, “Solving the energy efficient cover age problem in wireless sensor networks: A distributed genetic algorithm approach with hierarchical fitness evaluation,” Energies, vol. 11, no. 12, pp. 1–14, Dec. 2018.

[20] Z.-H. Zhan, Z.-J. Wang, H. Jin, and J. Zhang, “Adaptive distributed dif ferential evolution,” IEEE Trans. Cybern., vol. 50, no. 11, pp. 4633–4647, Nov. 2020

[21] Z. -H. Zhan et al., "Matrix-Based Evolutionary Computation," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 6, no. 2, pp. 315-328, April 2022.

[22] Akopov A S. MBHGA: a matrix-based hybrid genetic algorithm for solving an agent-based model of controlled trade interactions[J]. IEEE Access, 2025.