Development and analysis of adaptive mutation techniques in genetic algorithms

Zakhar Novikov[2], and *Evgenii* Sopov[1,2*]

[1]Reshetnev Siberian State University of Science and Technology, Institute of Informatics and Telecommunications, Krasnoyarsk, Russia
[2]Siberian Federal University, Institute of Space and Information Technology, Krasnoyarsk, Russia

Abstract. The proposed adaptive genetic algorithm demonstrates competitive performance, occasionally outperforming the standard approach. Although it shows promising adaptability, further refinements in parameter adjustments and hybridization with other evolutionary techniques could enhance its efficiency, making it a more powerful optimization tool.

1 Introduction

In computational optimization, a specific category of global optimization challenges is referred to as black-box problems. These tasks require identifying the best solution within a complex, often nonlinear search space where the underlying functions are unknown or not directly interpretable [1]. Overcoming these challenges is critical in many practical fields.

A black-box optimization problem can be described as follows: let $f(x_1, x_2, \ldots, x_n)$ be an objective function mapping a vector of input parameters $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ to a real value $f(x) \in R$. The objective is to identify a vector $\boldsymbol{x}^*$ that minimizes (or maximizes) the function:

$$f(\boldsymbol{x}^*) = min_x f(\boldsymbol{x}) \quad (1)$$

In real-world scenarios, it is often impossible to determine the exact global minimum (or maximum) due to factors such as measurement noise or numerical approximations. Therefore, this equality (1) is frequently replaced with an inequality to allow for some margin:

$$f(\boldsymbol{x}^*) - \varepsilon \leq min_x f(\boldsymbol{x}), \quad (2)$$

here ε represents an error tolerance, here set as $\varepsilon = 0.01$.

Solving black-box global optimization problems is challenging due to the complex, opaque nature of the objective function. Unlike traditional optimization methods that may depend on the structure of the problem, evolutionary algorithms – employed as metaheuristics – offer an effective strategy by providing a global search capability, exploring diverse areas of the solution space, and adapting to the uncertainties and complexities of the objective function.

Genetic algorithms (GA) are a type of evolutionary algorithm inspired by the principles of natural selection and genetics, uniquely suited for solving complex optimization problems [2]. In GA, candidate solutions are represented as individuals in a population, and each individual's performance is evaluated according to a fitness function, which reflects how well

* Corresponding author: evgenysopov@gmail.com

it solves the given problem. The algorithm then evolves this population through iterative cycles, aiming to optimize the fitness of the solutions without requiring knowledge of the objective function's form or gradient. This flexibility makes GAs particularly suitable for black-box optimization problems, where the underlying objective function is complex or undefined.

The strength of genetic algorithms lies in their use of evolutionary operators – selection, crossover, and mutation – which together foster both diversity and adaptability in the population. Selection favors individuals with higher fitness, while crossover and mutation introduce variation, allowing the algorithm to explore multiple solution paths and avoid premature convergence. One of the main advantages of GAs is their natural balance between exploring new areas of the solution space and refining promising regions, which is crucial in avoiding local optima and achieving a more global perspective on the search space. Additionally, genetic algorithms are particularly effective when dealing with variables that may be on different scales. This is why they often use binary encoding for variables, as opposed to algorithms designed specifically for continuous variables. The ability to handle such diverse data representations is a key reason why genetic algorithms are preferred over methods tailored only for real-valued variables.

Setting the parameters of an optimization algorithm, such as population size, crossover rate, and mutation probability, is essential for achieving efficient and effective search behavior. These parameters control the balance between exploration (searching new areas) and exploitation (focusing on promising regions) within solution space. In genetic algorithms, this balance is key to finding high-quality solutions, as parameters that are too aggressive may lead to chaotic, unfocused searches, while overly conservative setting risk converging too soon on suboptimal solutions.

One of the most challenging parameters to set is the mutation probability, which influences the introduction of diversity by altering individuals in the population. This parameter plays a crucial role in black-box optimization problems, where the underlying structure of the objective function is unknown, and different problem types benefit from different levels of mutation. The mutation probability is often represented as a dynamic value that can change from generation to generation, reflecting how much diversity needs to be introduced at each step of the optimization. Early in the process, higher mutation rates may be needed to explore the solution space broadly, while later generations may require lower mutation rates to refine promising solutions. This changing probability is central to our motivation for developing a new approach that evaluates diversity at the gene level and controls it through mutation management. By adjusting the mutation probability based on the diversity of the population, we can ensure a balance between exploration and exploitation, allowing for more effective and adaptive optimization.

2 Mutation probability configuration

2.1 Existing methods for mutation self-configuration

This section reviews various articles and studies that explore the use of mutation self-configuration in genetic algorithms. These methods dynamically adjust mutation rates throughout the algorithm's execution, based on various factors.

In article [3] a self-adaptive mutation approach based on chromosome similarity within the population is introduced. The key idea is to adjust the mutation probability dynamically depending on the similarity between chromosomes in each generation. The process involves two main steps: first, evaluating the similarity of the chromosomes in the population, and then adjusting the mutation probability based on this similarity. If the chromosome similarity

falls below a specified threshold, indicating high diversity, the mutation probability is reduced to prevent excessive mutation in a diverse population. Conversely, if the similarity increases beyond the threshold, signaling inbreeding, the mutation probability is increased to promote further diversity. This approach ensures that the mutation rate adapts according to the population's diversity, maintaining a balance between exploration and exploitation. This method works by comparing the similarity of chromosomes after applying the genetic operators (mutation and recombination). If the population shows too much diversity, the mutation probability is decreased, allowing for more convergence. If the population becomes too similar, the mutation probability increases to maintain diversity. The method's goal is to stabilize chromosome similarity around the predefined threshold as the generations evolve. This approach offers an adaptive mechanism to optimize the balance between exploration and exploitation in genetic algorithms.

Mutation scheme provided in [4] uses an adaptive approach where the mutation rate for each individual is determined by its fitness and its similarity to high-quality solutions in the population. The process begins by selecting a set of elite individuals that meet a fitness threshold, which are then used to construct a position weight matrix (PWM). This matrix represents the likelihood of each allele occurring at each gene site based on the successful individuals.

For each gene in the individual, a mutation rate is computed, taking into account two main factors:

1. The fitness of the individual being mutated. Higher fitness individuals typically have lower mutation rates to preserve their valuable genetic traits.
2. The alignment of the individual's alleles with the patterns observed in the PWM. If an individual's allele matches the patterns from high-quality solutions, the mutation rate is lower, as those genes are already well-optimized. On the other hand, if there is a deviation from the PWM pattern, the mutation rate increases to introduce variation.

This method allows for both exploitation and exploration in the search process: exploitation is achieved by carefully mutating high-fitness individuals to preserve good traits, while exploration occurs by applying higher mutation rates to less fit individuals, encouraging the discovery of new, potentially better solutions.

The two-stage adaptive genetic algorithm proposed in [5] addresses the limitations of canonical genetic algorithms by dynamically adjusting mutation and crossover probabilities based on the progress of optimization. Initially, the algorithm starts with fixed mutation and crossover probabilities. If the fitness of the best individual remains unchanged over several generations, the mutation probability is increased while the crossover probability is decreased. This adjustment introduces more randomness into the process, which helps the algorithm escape local optima.

If the stagnation persists, the mutation probability is further increased, and the crossover probability is further decreased, increasing the randomness even more. This step encourages exploration of new areas of the search space. Once the fitness of the best individual improves, indicating a better solution has been found, the mutation and crossover probabilities are reset to their initial values, focusing the algorithm on converging rapidly towards the new optimal solution.

Additionally, the algorithm adapts the search space size based on the convergence status, adjusting the parameters to balance the exploration and exploitation phases effectively. This adaptive strategy allows the algorithm to remain flexible, avoid premature convergence, and efficiently refine solutions when necessary.

2.2 Proposed method for mutation self-configuration

In this study, we introduce a dynamic mutation probability calculation method for genetic algorithms that adapts based on the distribution of gene values within each generation. Traditional approaches to mutation probability often rely on static or generation-dependent values; however, our approach calculates mutation probability at a more granular level, adjusting for each gene based on its distribution in the current population. This design aims to enhance the algorithm's adaptive capability, particularly when approaching local optima or final optimization stages, by providing more tailored mutation rates to individual genes.

The first step in our approach involves calculating individual probabilities for each gene position across the entire population. For each gene position i in the chromosome structure, the method evaluates the proportion of individuals in the population that have a value of *1* for that gene. Specifically, the gene probability for position i is derived by computing the ratio of individuals with a *1* at gene position i to the total number of chromosomes in the population:

$$p_i = \frac{\sum_{k=1}^{N} x_{k,i}}{N}, \tag{3}$$

here N is the total population size, and $x_{k,i}$ represents the value of gene i in the k-th chromosome. The gene-specific probability p_i, forms the basis for dynamically adjusting mutation rates.

To transform these probabilities into mutation rates, linear transformation function is applied. The idea is to map the probabilities p_i to a range defined by minimum and maximum mutation probabilities (P_{min} and P_{max} respectively). This mapping uses a function of the form:

$$P_i = P_{min} + 2|p_i - 0.5|(P_{max} - P_{min}). \tag{4}$$

This function translates the gene probability into a mutation probability P_i by scaling it proportionally between the predefined minimum and maximum values. The $2|p_i - 0.5|$ term ensures that genes with probabilities closer to *0.5* (indicating higher diversity) yield lower mutation rates, while those with probabilities nearer to *0* or *1* are more likely to mutate. This transformation method allows for a more adaptive mutation rate that ensures higher diversity within population.

The final mutation process involves applying the calculated mutation probabilities to each gene individually. For each gene i in a given chromosome, a mutation is executed with probability P_i.

The base method described above introduces a dynamic approach to calculating mutation probabilities, adapting based on the distribution of gene values within each generation. Following this, several alternative approaches are explored to modify the mutation process in different ways. These methods are designed to experiment with different strategies for mutation probability calculation and application:

1. **avg**: this modification introduces a method where the mutation probability is calculated as the average of individual gene probabilities across the chromosome. Rather than applying mutation rates that vary per gene, a single average mutation probability is applied uniformly to all genes in the chromosome. This approach simplifies the mutation process by reducing the need for individual gene-based mutation rates, making it easier to implement while still ensuring diversity through random mutations across the chromosome.
2. **reduction**: this modification introduces a function that adjusts mutation probabilities with a reduction factor over time. As the genetic algorithm progresses through generations, the contribution of the transformation function gradually decreases, scaling the mutation probabilities towards a target value. This is done by introducing a scaling factor based on the current generation relative to the total number of generations, which gradually reduces the effect of the transformation function as the algorithm nears its final stages. This approach aims to stabilize mutation rates over

time, potentially reducing excessive mutation in later generations and allowing for more refined convergence. Mutation probability calculates as:

$$P_{red\_i} = P_{target} + (P_i - P_{target}) * (1 - \frac{N_{cur}}{N_{max}}), \quad (5)$$

here P_{target} is the set target probability, N_{cur} is the number of current population, and N_{max} represents the maximum number of generation in this algorithm.

3. **diversity**: This modification introduces a method that adjusts mutation probabilities based on the diversity of the population between generations. The approach takes into account the change in diversity by comparing the current generation's diversity to that of the previous generation. If diversity decreases, the mutation probability is scaled up to encourage more exploration in search space. This is achieved through a scaling factor that reacts inversely to changes in diversity: as diversity decreases, the probability of mutation increases. This method helps maintain a balance between exploration and exploitation, ensuring that the algorithm adapts its mutation strategy based on the population's diversity. First relative diversity is calculated:

$$\Delta_{rel} = \frac{\Delta_i - \Delta_{i-1}}{\Delta_{i-1}}, \quad (6)$$

here Δ_i is a pairwise Euclidian distance between different chromosomes in i generation. Then Mutation probability calculates as:

$$P_i = P_{min} + 2|p_i - 0.5|(P_{max} - P_{min})\frac{1}{1+e^{\alpha\Delta}}, \quad (7)$$

that controls the rate at which the scaling factor decreases as the diversity changes between generations.

4. **iter**: This modification introduces an adaptive mutation strategy that switches between different mutation algorithms based on the progress of the algorithm. If the algorithm experiences no improvement for a certain number of iterations, the mutation approach is alternated. When the algorithm is using the standard mutation method, a fixed mutation probability is applied to the offspring chromosome. However, if the algorithm switches to a modified approach, a dynamic mutation method is used, where mutation probabilities are calculated individually for each gene. This alternating strategy aims to adapt the mutation process depending on the convergence of the algorithm, potentially enhancing exploration when progress stalls.

In the next section, these mutation strategies will be tested through numerical experiments to assess their performance and compare their effects on the algorithm's behavior.

3 Numerical experiments

3.1 Results of numerical experiments

This section details the testing parameters and comparative performance of the standard and modified algorithms under controlled settings. In our approach, we implemented random selection of individuals and used uniform crossover with a probability set at 0.25. For mutation rates, the standard algorithm applied a fixed mutation probability of 1/20. In contrast, the modified algorithm employed an adaptive mutation rate, with a minimum probability set at 1/60 and a maximum probability reaching 3/20.

The test functions chosen were Rastrigin function, Schwefel function, Ackley function and Rosenbrock function with ten variables, denoted as F1, F2, F3 and F4. To ensure accuracy, we conducted 40 runs for each algorithm. The outcomes of these experiments are presented in Table 1, Table 2, Table 3 and Table 4.

Table 1. Results for Ackey function (F3).

	Standard	Each	Avg	Reduction	Diversity	Iter
min	4.48	5.83	5.38	5.62	4.49	4.66
max	10.23	9.50	9.76	9.67	11.47	8.72
average	7.32	7.40	7.09	7.69	7.47	7.03

Table 2. Results for Rastrigin function (F1).

	Standard	Each	Avg	Reduction	Diversity	Iter
min	13.43	17.90	16.51	16.13	12.99	17.87
max	45.87	44.35	49.71	42.63	45.84	42.53
average	30.29	31.95	30.42	30.11	27.45	31.19

Table 3. Results for Schwefel function (F2)

	Standard	Each	Avg	Reduction	Diversity	Iter
min	306.96	362.63	499.09	254.12	377.29	435.19
max	1354.87	1292.26	1252.10	1414.80	1261.81	1267.97
average	849.41	927.59	903.51	866.45	786.97	899.96

Table 4. Results for Rosenbrock function (F4)

	Standard	Each	Avg	Reduction	Diversity	Iter
min	6.12	10.06	10.19	8.13	8.12	10.27
max	64.77	81.09	76.68	68.89	71.52	85.06
average	20.81	28.36	23.62	21.19	17.69	24.99

3.2 Discussion of numerical experiments

The proposed adaptive approach demonstrates competitive performance in comparison to the standard genetic algorithm, although it only occasionally surpasses the traditional method. This outcome suggests that the adaptive mechanism is beneficial but may need further refinement to consistently outperform the standard approach. The results indicate that, under specific conditions, the adaptive algorithm's ability to modify mutation rates enhances its capacity to explore and escape local optima. However, the gains achieved by the adaptive scheme are presently limited, occurring in a subset of test cases rather than across the board.

A notable advantage of the proposed method is its potential for flexibility and fine-tuning. By dynamically adjusting mutation rates in response to fitness stagnation, the algorithm better aligns with the current search demands, avoiding the rigidity of fixed parameters. This adaptability allows for more responsive exploration and exploitation phases in the evolutionary process, which, even if not universally superior, can prevent premature convergence in certain landscapes.

Unlike static mutation settings that may become suboptimal as the search progresses, this approach provides an element of self-regulation that reflects evolving search needs, an inherent advantage over fixed-rate algorithms.

Moreover, the potential for improvement is significant. Fine-tuning the adaptive ranges of mutation probabilities, or modifying how and when they adjust, could result in more consistent performance enhancements. By refining these parameters, it may be possible to make the algorithm more universally competitive across various optimization scenarios.

Despite its current limitations, the approach remains promising for further development, particularly for complex problems with diverse and challenging landscapes.

4 Conclusion

This work presents a novel adaptive genetic algorithm that dynamically adjusts mutation rates based on generational fitness trends. The approach, though only occasionally outperforming the standard genetic algorithm, demonstrates significant promise in its ability to adapt mutation intensity in response to optimization challenges. By altering mutation rates when improvement stagnates, our method helps mitigate common issues in genetic algorithms, such as premature convergence to suboptimal solutions.

While the adaptive algorithm has yet to consistently surpass its standard counterpart, it lays the groundwork for more responsive evolutionary strategies. The adaptability of the algorithm allows it to better balance exploration and exploitation by intensifying search efforts when trapped in local optima and maintaining stability when nearing potential solutions. These adaptive capabilities offer a clear advantage over fixed-parameter algorithms, which may not be as responsive to shifts in the optimization landscape.

Future enhancements could involve refining the adaptive thresholds, exploring additional dynamic parameters, and broadening the application of the adaptive framework to other types of evolutionary algorithms. The flexibility of adaptive mechanisms like ours shows promise for tackling complex, real-world problems where static parameters may not suffice, opening the door for further improvements in both efficiency and robustness.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant No.075-15-2022-1121).

References

1. L. M Rios, N. V. Sahinidis, D, Journal of Global Optimization, **56**, 1247-1293 (2013).
2. H. Holland, *Adaptation in Natural and Artificial Systems: An Intriductory Analysis with Application to Biology, Control, and Artificial Intelligence* (MIT press, Cambrige, 1975).
3. D. Smullen, A. Patel, J. Miller, and R. Ghosh, *Genetic algorithm with self-adaptive mutation controlled by chromosome similarity,* in IEEE Congress on Evolutionary Computation (CEC), 504-511 (2014).
4. F. Vafaee, G. Turán, P.C. Nelson, and T.Y. Berger-Wolf, *Among-site rate variation: adaptation of genetic algorithm mutation rates at each single site,* in Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 863-870 (2014).
5. H. Zhou, L. Wang, X. Li, and Y. Zhang. *A self-adaptive hybrid algorithm for planning city air terminals,* in IEEE Congress on Evolutionary Computation (CEC), 1-7 (2020).