Analysis of the Sensitivity of Genetic Algorithms to Parameter Settings: A Comparison of Fixed and Adaptive Approaches

Anna Krajčírová[1]

[1]Institute of Computer Science, Faculty of Philosophy and Science in Opava, Silesian University in Opava, Opava, Czech Republic

Abstract

Genetic algorithms (GAs) are effective optimization tools for complex, non-deterministic problems, where performance strongly depends on balancing exploration and exploitation via parameter selection. The performance of a genetic algorithm is influenced not only by the formulation of the fitness function but also depends on several algorithmic components, including population size, crossover probability, mutation probability, and the choice of genetic operators. Among these, mutation probability plays a key role in balancing exploration and exploitation during the search process. While fixed parameter settings are commonly used in practice, adaptive parameter control has the potential to improve algorithm performance by adjusting parameters according to the current state of the search.

This paper evaluates the impact of an adaptive mutation probability strategy while holding all other GA components constant. The adaptive approach is benchmarked against fixed mutation settings on three distinct problems: (1) a 1D continuous problem, (2) a 2D continuous problem, and (3) the Traveling Salesman Problem (TSP), a discrete combinatorial optimization problem. The experimental results are evaluated in terms of solution quality, convergence speed, and stability, highlighting the specific conditions under which adaptive mutation control provides a clear advantage.

Keywords

Genetic algorithms, parameter setting, parameter tuning, adaptive parameter control, optimization of a one-dimensional benchmark function, optimization o a two-dimensional benchmark function, the traveling salesman problem,

CEUR-WS.org/Vol-4323/paper25.pdf

1. Introduction

Genetic algorithms (GAs), a class of evolutionary algorithms inspired by natural selection, originated in the 1960s with Rechenberg's Evolution Strategies and were formalized by Holland in the early 1970s [1]. They are widely applied to complex optimization problems—such as combinatorial or non-linear tasks with numerous local optima—where classical gradient-based methods fail and exact solutions are computationally infeasible.

A major strength of GAs is their ability to search large, unstructured solution spaces without prior domain knowledge. However, their performance heavily depends on proper parameter tuning (e.g., population size, crossover rate, and mutation rate). Suboptimal parameter settings can cause premature convergence, algorithm stagnation, or insufficient exploitation of promising regions [2].

While traditional GAs rely on fixed parameters set prior to execution, adaptive parameter control dynamically modifies values during the run based on the search status. This approach dynamically balances exploration (searching new regions) and exploitation (refining existing high-quality solutions), thereby improving overall robustness.

This paper experimentally evaluates fixed versus adaptive mutation probability control under otherwise identical GA configurations. The comparison is conducted across three distinct benchmark problems: two continuous optimization tasks of varying complexity and the Traveling Salesman Problem (TSP) as a discrete combinatorial task. The benchmark suite serves to analyze how search-space characteristics influence the effectiveness of adaptive mutation control.

ITAT'26: Information Technologies – Applications and Theory, September 25–29, 2026, Biele Karpaty, Slovakia

anna.krajcirova@fpf.slu.cz (A. Krajčírová)

0009-0009-6509-4021 (A. Krajčírová)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-09

2. Genetic Algorithms

GAs mirror biological evolution by evolving a population of candidate solutions that gradually adapt to their environment. In computational terms, each individual represents a potential solution encoded as a chromosome.

During evolution, genetic operators—selection, crossover, and mutation—govern parent selection, information recombination, and random variation to form new generations. Solution quality is guided by a *fitness function*, which assigns each individual a scalar value reflecting its suitability. Higher-fitness individuals are preferentially selected for reproduction, driving the population toward optimal solutions over successive generations [3].

2.1. Algorithm Scheme

The basic principle of a genetic algorithm can be described by the following pseudocode:

```
initialize population P
evaluate the fitness of each individual in P

while (termination condition not satisfied):
    create a new population P'

    preserve the best individuals (elitism)

    while (|P'| < population size):
        select parents p1, p2 (selection)
        create offspring (crossover)
        apply mutation
        add offspring to P'

    replace P ← P'
    evaluate fitness

return the best solution found
```

Typical termination criteria include reaching a predefined maximum number of generations or finding a sufficiently high-quality solution [4, 1].

2.2. Solution Representation

The representation of an individual depends on the type of problem being solved. Common approaches include:

- **Binary representation**, where a solution is encoded as a bit string. This is the classical approach used in early genetic algorithms.
- **Real-valued representation**, where individuals are represented as vectors of real numbers. This representation is commonly used for continuous optimization problems; in our case, 1D and 2D benchmark functions.
- **Permutation representation**, which is typically employed for combinatorial optimization problems such as the traveling salesman problem (TSP). In this case, the chromosome represents the order in which elements are visited.

The choice of an appropriate representation has a significant impact on the effectiveness of a genetic algorithm, as it influences both the design of genetic operators and the algorithm's ability to explore the search space efficiently [2].

2.3. Genetic Operators

The exploration of the search space in genetic algorithms is performed through the genetic operators introduced above: selection, crossover, and mutation.

2.3.1. Selection Operator

The selection operator determines which individuals will be chosen as parents for the creation of a new generation. Its objective is to favor higher-quality individuals, i.e., those with better fitness values, while maintaining sufficient population diversity.

This study employed the *tournament selection* method. In this approach, a subset of individuals is randomly sampled from the population, and the fittest individual within this group is selected to become a parent. This process is repeated until a sufficient number of parents is selected to form the next generation. The primary advantages of tournament selection are its computational simplicity and the capability to adjust the selection pressure by varying the tournament size. Generally, a larger tournament size exerts greater selection pressure; while this accelerates convergence, it may lead to a loss of population diversity and cause the algorithm to become prematurely trapped in a local optima [5, 6].

Another commonly used method is *Roulette Wheel Selection*, where the probability of selecting an individual is proportional to its fitness relative to the total fitness of the population. An alternative approach is *rank-based selection*, which assigns selection probabilities according to the ranking of individuals rather than their absolute fitness values, thereby reducing the influence of extreme fitness values [4, 6, 7].

2.3.2. Crossover Operator

The crossover operator is used to combine the genetic information of two parent individuals in order to create offspring. This operator enables the exploration of new regions of the search space.

For continuous optimization problems, *arithmetic crossover* was employed, where the offspring is generated as a linear combination of the parents:

$$\text{offspring} = \alpha p_1 + (1 - \alpha) p_2 \tag{1}$$

For the traveling salesman problem, the *Order Crossover* (OX) operator was used. This operator preserves the relative order of elements and always produces valid permutations. Specifically, OX constructs offspring by copying a continuous slice of cities from one parent and filling the remaining positions with elements from the second parent in the order they appear, skipping any cities already present.

Other commonly used crossover operators include one-point and two-point crossover, which are typically applied to binary representations, and *uniform crossover*, where genes are inherited according to a randomly generated mask. For permutation-based problems, operators such as *Partially Mapped Crossover* (PMX) are frequently employed to preserve permutation validity [4].

2.3.3. Mutation Operator

Mutation introduces random modifications into individuals and serves to maintain population diversity. It also helps the algorithm escape from local optima.

For continuous optimization problems, mutation was implemented by adding Gaussian noise to individual variables. For the TSP, *swap mutation* was applied, in which two randomly selected elements of the permutation are exchanged.

Other frequently used mutation operators include *uniform mutation*, which replaces a gene value with a randomly selected value from its domain, and *non-uniform mutation*, where the magnitude of mutations gradually decreases during the evolutionary process. In permutation-based problems,

inversion mutation, which reverses the order of elements within a selected segment of the chromosome, is also commonly used [4].

2.4. GA Parameters

The performance of a genetic algorithm depends on its parameters; in our case:

- population size,
- crossover probability P_c,
- mutation probability P_m,
- number of generations.

These parameters influence the balance between exploration (searching new regions of the search space) and exploitation (improving already discovered solutions) [1].

As we can see in Figure 1, genetic algorithms can be divided into two main categories based on how their parameters are controlled. The traditional approach, commonly referred to as *parameter tuning*, relies on fixed parameter values specified before the algorithm starts. The second category consists of algorithms with adaptive parameter control.

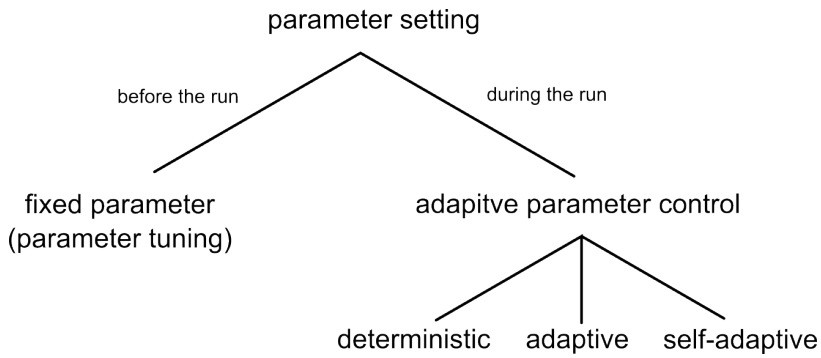


Figure 1: Various approaches to parameter control in genetic algorithms [8].

Adaptive parameter control methods can be further divided into three groups:

- **Deterministic control**, where parameter values are modified during the execution of the algorithm according to predefined rules. For example, the mutation probability may decrease linearly over time, promoting exploration during the early stages of the search and exploitation in later stages.
- **Adaptive control**, where parameter values are adjusted based on the current state and performance of the search process. A feedback mechanism determines when and how the parameters should be modified.
- **Self-adaptive control**, where the parameters themselves are encoded directly into the chromosomes of individuals and evolve together with the candidate solutions. The evolutionary process implicitly determines which parameter settings provide the greatest evolutionary advantage, allowing each individual to maintain its own parameter values.

3. Experimental Setup

The objective of the experiments was to analyze the influence of genetic algorithm parameter settings on solution quality, convergence speed, and result stability. Particular attention was paid to the effect of the mutation probability (P_m). In addition to investigating the impact of parameter settings, we compared the conventional approach using fixed parameters with an adaptive strategy for controlling the mutation probability.

3.1. Benchmark Problems

Three benchmark problems were selected to represent different classes of optimization tasks. The first two problems are continuous optimization problems with increasing complexity and are intended to evaluate the influence of parameter settings within the same problem domain. The third benchmark, the traveling salesman problem, was intentionally chosen as a discrete combinatorial optimization problem to investigate whether the observed effects of parameter settings remain consistent across fundamentally different search spaces.

The objective of the benchmark suite is not to compare the optimization problems themselves, but to investigate how the effectiveness of different parameter settings varies across problems with different search-space characteristics.

3.1.1. One-Dimensional Optimization Problem

As a one-dimensional optimization problem, we considered the function

$$y = \sin(x) \cdot x \tag{2}$$

defined on the interval $x \in \langle 0, 20 \rangle$.

This problem serves as a basic benchmark scenario with a relatively simple fitness landscape. Our goal was to test whether and how different algorithm parameter settings would affect such a simple task.

3.1.2. Two-Dimensional Optimization Problem

The two-dimensional optimization problem was defined by the function

$$z = \sin(x) \cdot \cos(y) + 0.1x + 0.1y \tag{3}$$

defined on the domain $(x, y) \in \langle 0, 20 \rangle \times \langle 0, 20 \rangle$.

This function combines a periodic component with a linear trend, creating a complex fitness landscape containing multiple local optima. The problem is therefore suitable for evaluating the algorithm's ability to escape local optima and effectively explore the search space.

3.1.3. Traveling Salesman Problem (TSP)

The third benchmark problem, traveling salesman problem (TSP), complements the previous continuous benchmark problems by representing a discrete NP-hard optimization task. The objective of the TSP is to find the shortest possible route that visits all given cities exactly once and returns to the starting city. The TSP is a well-known NP-hard optimization problem, meaning that the number of possible solutions grows exponentially with the number of cities. Even for a relatively small instance with $n = 10$ cities, the number of possible tours is 181,440. Consequently, exhaustive search becomes impractical for larger problem instances, making the TSP a suitable benchmark for genetic algorithms [9].

In this study, an instance containing $n = 30$ cities was considered, resulting in approximately 4.42×10^{30} possible tours. The city coordinates were generated randomly in a two-dimensional plane. To ensure reproducibility, the coordinates were generated only once and subsequently used in all experimental runs [6, 10]. We can see the map of the cities in the Figure 2.

The fitness function was defined as the reciprocal of the total length d of the closed tour:

$$f = \frac{1}{d}. \tag{4}$$

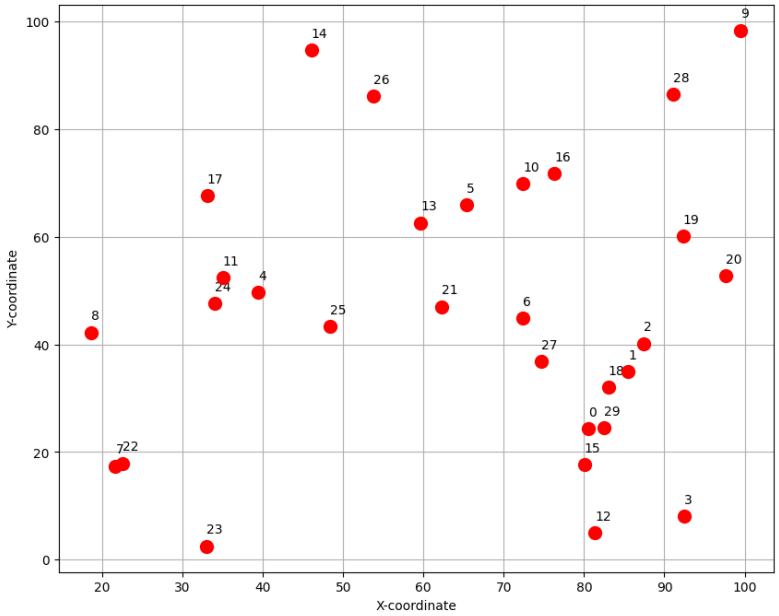


Figure 2: The map of randomly generated cities.

3.2. Genetic Algorithm Configuration

The following genetic algorithm configuration was used in all experiments:

- population size: 80 and 180 individuals,
- number of generations: 30 for the one-dimensional problem, 150 for the two-dimensional problem, and 400 for the TSP,
- crossover probability: $P_c = 0.8$ and $P_c = 0.5$,
- mutation probability: $P_m = 0.01$, $P_m = 0.05$, and $P_m = 0.1$,
- selection method: tournament selection (tournament size of 3),
- elitism: the best individual was always preserved in the next generation.

The primary objective of the experiments was to compare fixed and adaptive mutation probability control using the baseline configuration ($pop_size = 80$, $P_c = 0.8$). Additional experiments with a larger population ($pop_size = 180$) and a lower crossover probability ($P_c = 0.5$) were included only as reference configurations. Their purpose was not to perform a comprehensive parameter study, but rather to verify whether the observed trends remained consistent under different genetic algorithm settings.

For continuous optimization problems, continuous variables were mutated by adding zero-mean Gaussian noise with a standard deviation set empirically to 0.5 (representing 10% of the domain range). Crossover for these problems was performed using the arithmetic crossover operator with the randomly generated alpha within the interval $(0; 1)$. In contrast, for the TSP, a swap mutation was applied alongside the Order Crossover (OX) operator.

Each algorithm configuration was evaluated using 30 independent runs.

3.3. Adaptive Parameter Control

In the adaptive variant of the genetic algorithm, the mutation probability P_m was dynamically adjusted based on optimization progress to resolve stagnation while enabling the fine-tuning of promising solutions.

When the best fitness value improved in a given generation, P_m was multiplied by a factor of 0.9. This gradual reduction in randomness shifted the search toward exploitation in the vicinity of high-

quality solutions. Exploration was preserved through crossover, continuous Gaussian perturbations, and predefined lower bounds on P_m.

Conversely, if no improvement occurred for 10 consecutive generations, the algorithm was considered stagnating and P_m was increased. For the continuous 1D and 2D problems, a multiplication factor of 1.5 was applied to rapidly restore population diversity and explore new search-space regions. For the discrete TSP, a lower factor of 1.1 was used to prevent aggressive mutations from disrupting existing high-quality tour components during its longer evolutionary process.

To maintain stability, P_m was constrained within predefined lower and upper bounds throughout the run. All adaptation constants were chosen empirically to provide smooth parameter adjustments; their systematic optimization remains a topic for future work.

4. Experimental Results

This section presents and evaluates the data obtained from the experiments. For each benchmark problem, a table containing the basic statistical results is provided together with four figures illustrating:

- the convergence rate for different values of P_m,
- the convergence rate for the same P_m but different values of P_c and population size (*pop_size*),
- the variability of the generation in which the best solution was found (*best_gen*) for different values of P_m,
- the variability of *best_gen* for the same P_m but different values of P_c and *pop_size*.

To simplify the notation of individual algorithm configurations, the following naming convention is used: *Problem-Variant-pop_size* $\times P_c \times P_m$

For example, a genetic algorithm solving the one-dimensional problem with fixed parameters, population size $pop_size = 80$, crossover probability $P_c = 0.8$, and mutation probability $P_m = 0.01$ is denoted as: *1Dfix80x0.8x0.01*.

4.1. One-Dimensional Optimization Problem

This subsection evaluates the performance of the genetic algorithm on the one-dimensional optimization problem. Both fixed-parameter and adaptive variants are compared. The results are summarized in Table 1 and supplemented by convergence and variability plots.

Table 1
Comparison of GA parameter configurations for the one-dimensional optimization problem.

Alg.	Mean fit	Std. fit	Med. best_gen	Std. best_gen
fix80x0.8x0.01	20.4447831	5.1E-06	14	4.88
fix80x0.8x0.05	20.4447826	7.9E-06	15	2.02
fix80x0.8x0.1	20.4447841	7.2E-15	15	2.42
adapt80x0.8x0.01	20.4447841	7.2E-15	14	4.12
adapt80x0.8x0.1	20.4447841	7.2E-15	15	2.37
fix180x0.8x0.01	20.4447841	7.2E-15	14	1.19
fix80x0.5x0.01	20.4447840	2.8E-07	15	3.02

The results show that all tested genetic algorithm configurations achieve virtually identical fitness values (≈ 20.44478), corresponding to the global optimum of the objective function. Consequently, the differences between the configurations are reflected primarily in convergence speed and stability, although these differences remain relatively small compared with those observed in more complex benchmark problems.

As can be seen from Table 1 and Figure 3, the influence of the mutation probability is minimal for this problem due to its relatively simple fitness landscape. Similarly, adaptive parameter control does not provide any significant improvement over fixed parameter settings.

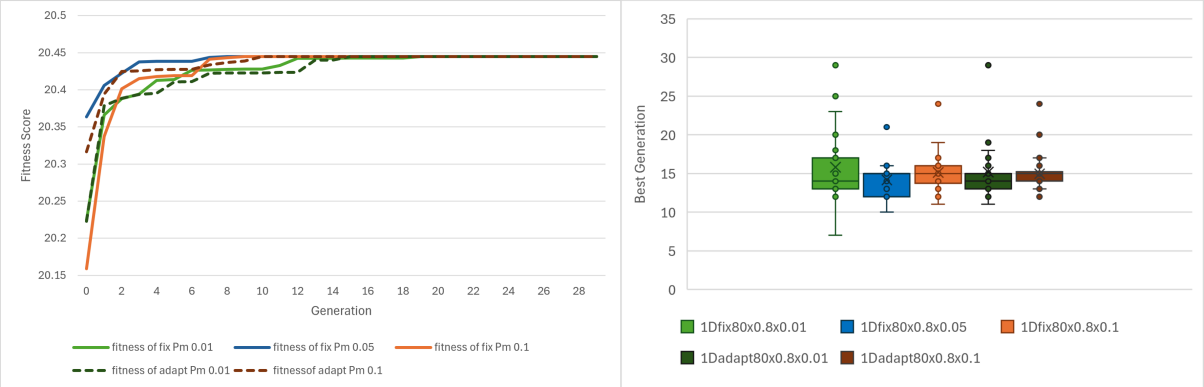


Figure 3: Left: Comparison of convergence speed and solution quality for the *1Dfix80x0.8xPm* and *1Dadapt80x0.8xPm* configurations. Right: Comparison of the variance of *best_gen* for the *1Dfix80x0.8xPm* and *1Dadapt80x0.8xPm* configurations.

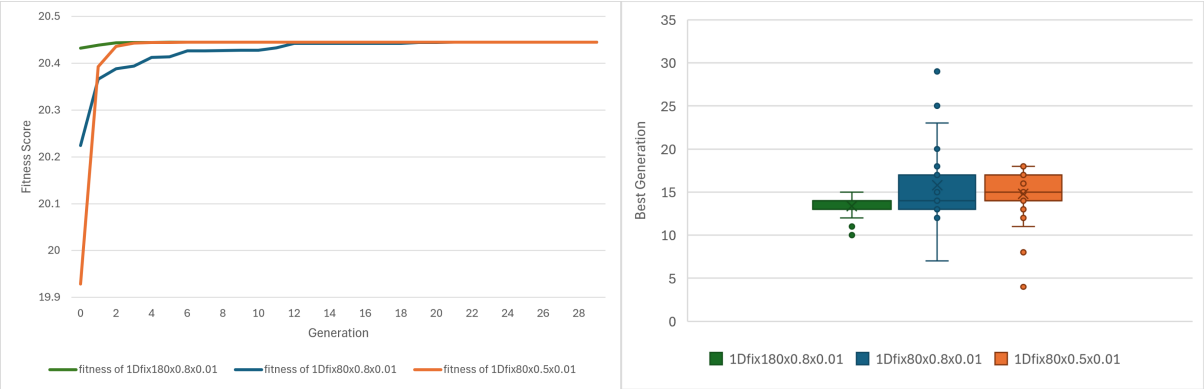


Figure 4: Left: Comparison of the variance of *best_gen* for the *1Dfix80x0.8xPm* and *1Dadapt80x0.8xPm* configurations. Right: Comparison of convergence speed and solution quality for configurations with $P_m = 0.01$.

From the convergence perspective, the best performance is achieved by the configuration with the larger population size (*1Dfix180x0.8x0.01*), which exhibits the lowest variability (*Std. best_gen = 1.19*) while maintaining a low median convergence generation. In contrast, a lower crossover probability (*1Dfix80x0.5x0.01*) results in slower and less stable convergence, as illustrated in Figures 4.

4.2. Two-Dimensional Optimization Problem

This subsection presents the results obtained for the two-dimensional optimization problem, which represents a more complex fitness landscape with a larger number of local optima. A summary of the results is provided in Table 2 and complemented by the corresponding figures.

Table 2
Comparison of GA parameter configurations for the two-dimensional optimization problem.

Algorithm	Mean fit	Std fit	Med best_gen	Std besst_gen
fix80x0.8x0.01	6.2733	0.3260	37	42.41
fix80x0.8x0.05	6.3052	0.2894	36.5	42.32
fix80x0.8x0.1	6.2121	0.2331	33	42.08
adapt80x0.8x0.01	6.2765	0.2375	131.5	49.89
adapt80x0.8x0.1	6.3323	0.2876	112.5	45.62
fix180x0.8x0.01	6.4568	0.2523	30	27.78
fix80x0.5x0.01	6.2491	0.2332	36.5	38.02

Unlike the one-dimensional problem, the differences between individual configurations become much more pronounced in this case. As shown in Table 2 and Figure 6, the highest average fitness value is achieved by the configuration with the larger population size (*2Dfix180x0.8x0.01*), highlighting the importance of population diversity when solving more complex optimization problems.

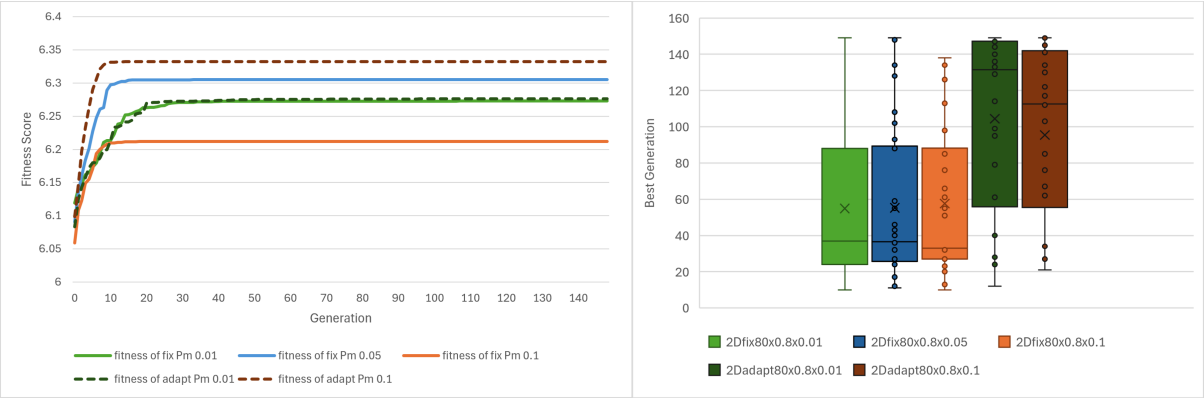


Figure 5: Left: Comparison of convergence speed and solution quality for the *2Dfix80x0.8xPm* and 2Dadapt80x0.8xPm configurations. Right: Comparison of the variance of *best_gen* for the 2Dfix80x0.8xPm and *2Dadapt80x0.8xPm* configurations.

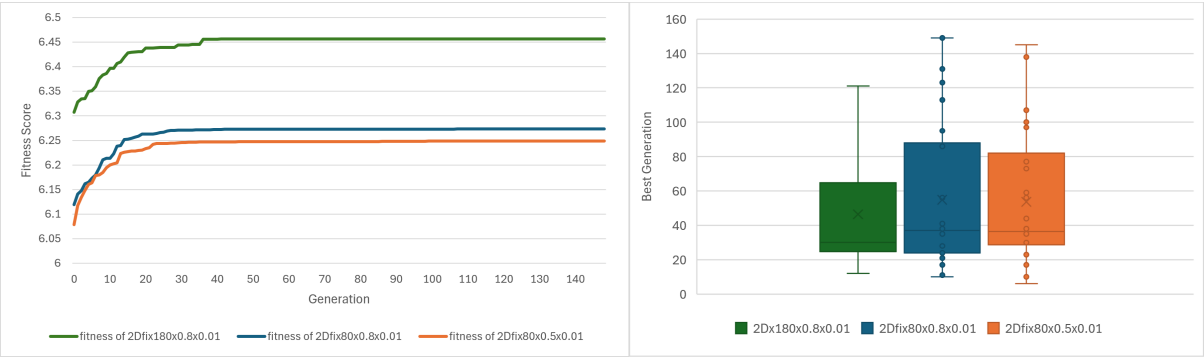


Figure 6: Left: Comparison of convergence speed and solution quality for configurations with $P_m = 0.01$. Right: Comparison of the variance of *best_gen* for configurations with $P_m = 0.01$.

A higher mutation probability ($P_m = 0.1$) results in a decrease in solution quality, indicating an excessive level of randomness in the search process. In contrast, the intermediate mutation probability ($P_m = 0.05$) provides a better balance between exploration and exploitation. This configuration also exhibits good stability, as illustrated in Figure 5 (right).

Compared with the corresponding fixed-parameter configurations, the adaptive genetic algorithm achieves slightly higher fitness values. This improvement is accompanied by significantly slower convergence, with the median generation of the best solution reaching up to 131.5. The results indicate that adaptive parameter control may encourage a more extensive exploration of the search space, although at the expense of convergence speed.

Consequently, when highly accurate solutions are not strictly required and an approximate solution suffices, an adaptive algorithm may be unnecessarily resource-intensive.

4.3. Traveling Salesman Problem

This subsection focuses on the application of the genetic algorithm to the traveling salesman problem, a discrete combinatorial optimization problem. The results are summarized in Table 3 and complemented by the corresponding figures. To facilitate interpretation, the quality of a solution is evaluated using the total route length (*distance*) rather than the fitness value itself.

Table 3
Comparison of GA parameter configurations for the traveling salesman problem.

Algorithm	Mean dis	Std dis	Med best_gen	Std best_gen
fix80x0,8x0,01	576,69	45,47	365	59,33
fix80x0,8x0,05	535,90	38,49	327,5	57,35
fix80x0,8x0,1	521,00	28,23	263,5	99,01
adapt80x0,8x0,01	546,16	47,10	360	31,02
adapt80x0,8x0,1	504,60	27,92	333,5	85,12
fix180x0,8x0,01	510,06	36,55	322	93,08
fix80x0,5x0,01	613,19	40,44	349	44,88

For the traveling salesman problem, the differences between configurations are the most pronounced. The best overall performance is achieved by the adaptive genetic algorithm (*TSPadapt80x0.8x0.1*), which minimizes the average route length to 504.60, as shown in Table 3 and Figure 7.

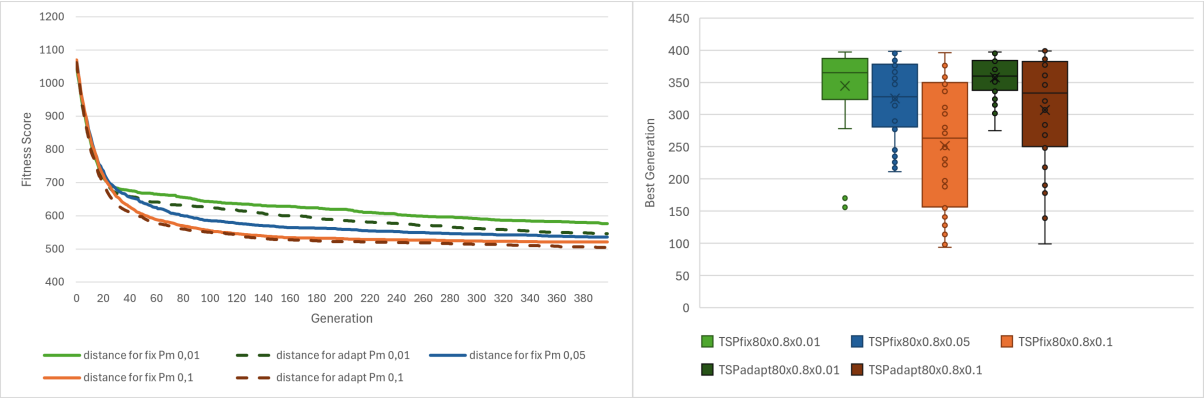


Figure 7: Left: Comparison of convergence speed and solution quality for different values of P_m. Right: Comparison of the variance of *best_gen* for the *TSPfix80x0.8xPm* and *TSPadapt80x0.8xPm* configurations.

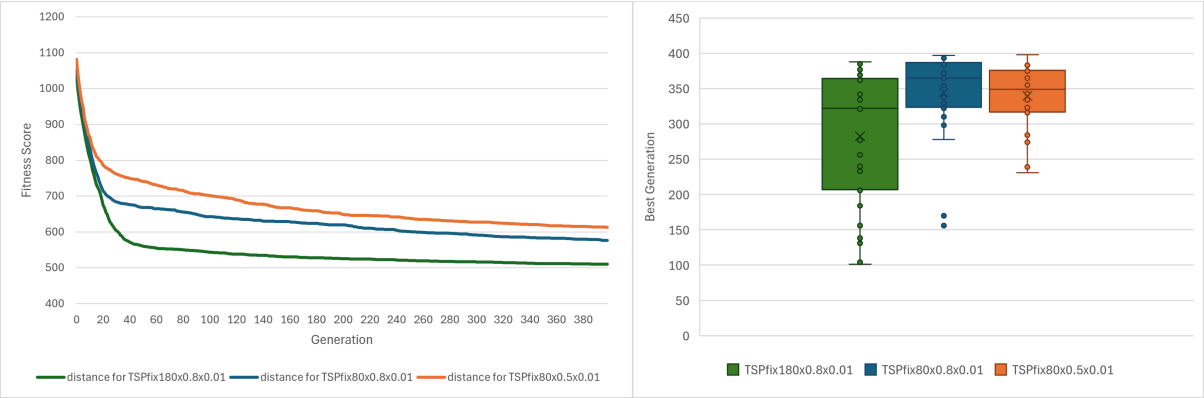


Figure 8: Left: Comparison of convergence speed and solution quality for configurations with $P_m = 0.01$. Right: Comparison of the variance of *best_gen* for configurations with $P_m = 0.01$.

Interestingly, the improvement achieved by *TSPadapt80x0.8x0.1* over *TSPfix80x0.8x0.1* is smaller than the improvement observed between *TSPadapt80x0.8x0.01* and *TSPfix80x0.8x0.01*. This behavior contrasts with the trend observed in the two-dimensional optimization problem, where the opposite effect was observed.

A higher mutation probability ($P_m = 0.1$) proves beneficial for the TSP, as it helps the algorithm escape local optima more effectively. In contrast, a low mutation probability leads to inferior results, as demonstrated by the *TSPfix80x0.8x0.01* configuration.

Another interesting observation is the apparent inverse relationship between the mutation probability and the standard deviation of the obtained tour lengths. Configurations with higher mutation probabilities consistently exhibited lower values of Std dis. This behavior suggests that stronger mutation promotes a more consistent exploration of the search space, allowing independent runs to converge to solutions of more similar quality instead of becoming trapped in different local optima. The same tendency can be observed for the adaptive configurations.

Increasing the population size (*TSPfix180x0.8x0.01*) also improves solution quality, although at the cost of higher convergence variability. Conversely, reducing the crossover probability (*TSPfix80x0.5x0.01*) negatively affects both solution quality and convergence speed, as can be seen in Table 3 and Figures 8.

Overall, the adaptive approach appears to be the most effective strategy for this problem, as it can dynamically respond to stagnation and achieve a better balance between exploration and exploitation.

5. Conclusion

The experimental results confirm that the influence of genetic algorithm parameter control strongly depends on problem complexity and its characteristics. While simple tasks can be solved effectively using fixed parameters, adaptive parameter control becomes increasingly vital as complexity grows, aligning with the fundamental exploration–exploitation trade-off [11, 8].

For the 1D problem, all configurations reached the global optimum with minimal differences in convergence speed and stability, rendering adaptive control unnecessary due to the simple fitness landscape. In the 2D problem, parameter settings had a more pronounced effect; a moderate mutation probability offered the best balance, whereas higher values degraded quality. Here, the adaptive approach achieved slightly better fitness than fixed-parameter setups by exploring the search space more extensively, albeit at the cost of slower convergence.

The Traveling Salesman Problem (TSP) yielded the most significant performance gains. Adaptive control achieved the shortest average tour length by dynamically increasing mutation probability to overcome stagnation and prevent premature convergence.

An interesting observation concerns the dependence of adaptive parameter control on the initial mutation probability. For the traveling salesman problem, the adaptive approach provided greater improvement when the initial mutation probability was low, whereas the benefit was less pronounced for higher initial mutation probabilities. This behavior can be explained by the adaptive mechanism's ability to overcome premature convergence through an increase in population diversity.

In contrast, the opposite trend was observed for the continuous two-dimensional optimization problem, where the adaptive approach provided greater improvement when the initial mutation probability was higher. In this case, the adaptive mechanism gradually reduced the mutation probability as the population approached the optimum, enabling the search process to transition smoothly from exploration to exploitation. These findings demonstrate that the effectiveness of adaptive parameter control is closely related to the characteristics of the optimization problem.

Across all problems, larger populations improved solution quality and stability at the cost of higher computational effort, whereas lower crossover probabilities generally degraded performance. Nevertheless, it was possible to observe the reference configurations followed the same overall trends as the baseline configuration. While these experiments were included only for comparison, they support the conclusion that the observed behavior of the adaptive mutation control is consistent across the considered genetic algorithm configurations.

Overall, adaptive parameter control is particularly advantageous for complex, multimodal optimization problems, while fixed settings remain sufficient and computationally efficient for simple tasks [11, 8]. Future work could conduct sensitivity analyses on the adaptation factors, evaluate alternative genetic operators (e.g., tournament vs. roulette-wheel selection [6]), and extend the benchmark suite with rigorous statistical significance testing.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT and DeepL to translate the text from Czech to English. After using these tool(s)/service(s), the author reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. V. T. K. Trinh, Research on genetic algorithms: Concepts, models, and applications, International Journal of Education and Social Science Research (IJESSR) 8 (2025) 354–373.

[2] K. Le, T. Wei, J. Tagalogon, Genetic algorithm and application, Genetic algorithm and Application (2023). URL: https://www.scienceopen.com/. doi:10.14293/pr2199.000533.v1.

[3] J. Koza, Genetic programming as a means for programming computers by natural selection, Statistics and Computing 4 (1994). doi:10.1007/bf00175355.

[4] H. Lingaraj, A study on genetic algorithm and its applications, International Journal of Computer Sciences and Engineering 4 (2016) 139–143.

[5] B. Miller, D. E. Goldberg, Genetic algorithms, tournament selection and the effects of noise, 1995.

[6] N. Razali, J. Geraghty, Genetic algorithm performance with different selection strategies in solving tsp, volume 2, 2011.

[7] P. BAJPAI, M. Kumar, Genetic algorithm - an approach to solve global optimization problems, Indian Journal of Computer Science and Engineering 1 (2010) 199–206.

[8] A. Eiben, Z. Michalewicz, M. Schoenauer, J. Smith, Parameter control in evolutionary algorithms, in: F. G. Lobo, C. F. Lima, Z. Michalewicz (Eds.), Parameter Setting in Evolutionary Algorithms, volume 54 of *Studies in Computational Intelligence*, Springer Verlag, 2007, pp. 19–46. URL: https://inria.hal.science/inria-00140549. doi:10.1007/978-3-540-69432-8, http://www.springerlink.com/content/978-3-540-69431-1/.

[9] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, A. Schrijver, P. Sayer, Combinatorial optimization, John Wiley & Sons Incorporated, 1997.

[10] R. Matai, S. Singh, M. Mittal, Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches, 2010. doi:10.5772/12909.

[11] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, Springer, 2003.