Hyperparameter optimization in Support Vector Machine classifier using genetic algorithm, microbial genetic algorithm and simulated annealing techniques

Andres Sebastián Salazar Alturo

Lecturer:
Dr Chris Jhonson

University of Sussex
Artificial Intelligence and adaptive systems MSc
Brighton, UK
2023

Abstract:

Hyperparameter tuning is key to obtaining the best results using machine learning techniques. If the tuning is done properly in a machine learning algorithm, a noteworthy higher accuracy can be obtained. Furthermore, hyperparameters can be set by heuristic techniques or based on related work. However, choosing the appropriate algorithm to the problem configuration takes time and computer resources, in addition each model must be tested multiple times with different sets of hyperparameters as not all data sets have the same characteristics, thus require different hyperparameter combinations to compare and select the one with the best performance. Therefore, to avoid the tedious process of tuning the hyperparameters manually. The use of a genetic algorithm (GA), microbial GA and simulating annealing (SA) are proposed to optimize the hyperparameters of the machine learning model Support Vector Machine (SVM) using the data set MNIST. The main goal of this project is to compare the three optimization algorithms and identify which achieves the highest accuracy. The performances will be measured using F1-Score evaluation metric.

Keywords: Machine learning, hyperparameter optimization, optimization techniques, genetic algorithms, Support Vector Machine classifier.

Contents

List of figures

List of tables

1 Introduction:

In XXI century, the machine learning models have been used in a wide range of industries, from social media to detect fake Instagram profiles using supervised machine learning algorithms (Purba, Asirvatham , & Murugesan, 2020) to product sentiment analysis in Amazon (Wassan, Chen, Shen, Waqar, & Jhanjhi, 2021).

Moreover, a machine learning algorithm with the correct hyperparameter can significantly increase the model's performance. Hence, the hyperparameter tuning has been an important step in machine learning experimentation and genetic algorithms have been use to optimize the hyperparameters to detect transaction frauds (Tayebi & El Kafhali, 2021) and Arabic sentiment analysis (Elgeldawi, Sayed, Galal, & Zaki, 2021).

Additionally, (Alibramhim & Ludwig, 2021) carry through a different approach, where compared three different algorithms to find the best hyperparameters for a neural network. The algorithms were Grid Search, Bayesian Algorithm and Genetic Algorithm. The authors conclude that there was not a high difference between them. However, the generic algorithm had a better performance than the grid search and the Bayesian algorithm.

Similarly, (Aszemi & Dominic, 2019) tested a genetic and Bayesian algorithms to extract automatically the best hyperparameters configuration in a convolutional neural network. The author concludes that using genetic algorithms is computationally cost and recommends running the algorithm in multiple GPUs and the grid search is impractical as there are millions of possible combinations.

On the other hand, (Ali, Awwad, Al-Razgan, & Maarouf, 2023) compared a large variety of algorithms for a hyperparameter search in machine learning algorithms to optimize the computational complexity. In contrast to (Aszemi & Dominic, 2019), concludes: *"The time complexity of the genetic algorithm was reduced when compared with the time complexity of other algorithms".* What proves the viability of the experimentation in this report.

The above related work addresses the question that is solved by the experimentation: What optimization algorithm between the three proposed (Genetic algorithm, Microbial GA, and Simulated annealing) have the best performance (measured in F1 score metric) optimizing the hyperparameters of a SVM classifier?

Setting the best hyperparameters in a machine learning model is key to reach the best possible classification models. However, due to the differences in every data set and classification algorithms, the hyperparameters change and take time and computer resources to find the optimal parameters. Hence, the importance of developing, implementing, comparing, and evaluating the three proposed optimization algorithms.

2 Methods:

Proposed methodology:

The proposed optimization algorithms are: Simulated annealing, Genetic algorithm, and Microbial Genetic algorithm. The SVM vector machine classifier was selected as classification model due to its high performance in classification tasks (Liang, 2004). Moreover, to determine the best optimization algorithm in hyperparameter tuning task, these algorithms were compared in terms of their F1 score as fitness. Figure 1 shows the workflow of the experiment. On the other hand, the hyperparameters chosen to be evolved are: Kernel, C, Degree, Gamma and Coef0.

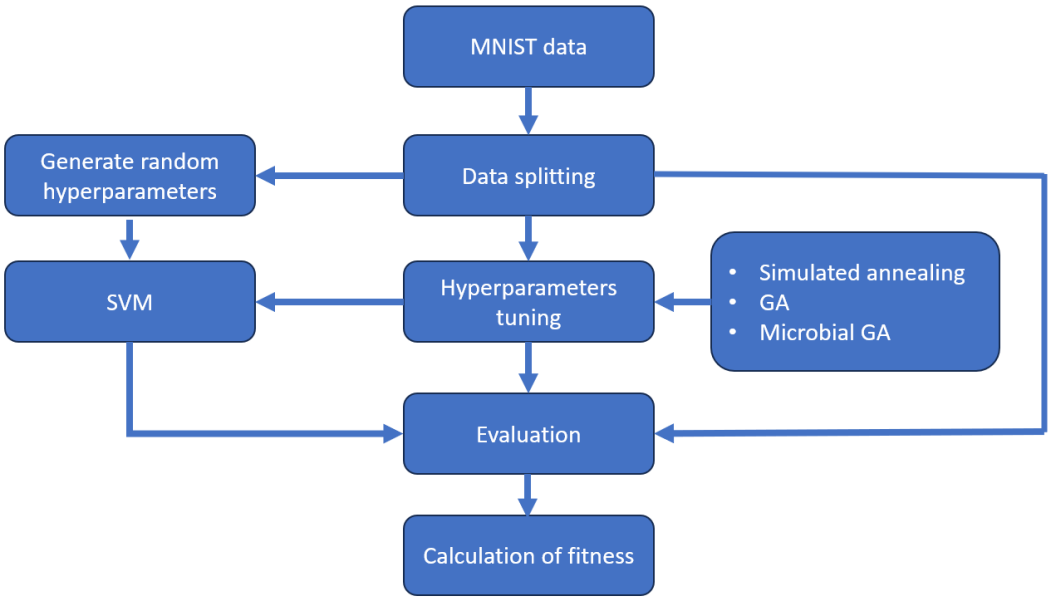


Figure 1. Experiment workflow proposed.

MNIST data set:

The MNIST data set is a subset of a larger set available from NIST. The digits are size-normalized and centered in a fixed-size image. MNIST is composed of 60,000 handwritten digits where 10,000 are test examples (LeCun, Cortes, & Burges). The data was extracted from *Kaggle*.

It was chosen as needs minimal preprocessing and formatting, so it was possible to work consistently in the optimization methods.

From the original data set there were randomly generated the index of the digits used in the classification model. As the data set is large and the computer complexity is high, it was decided to use 5,000 examples from where 4,000 were used as training data and the remaining 1,000 as test data. Figure 2 provides an example of digits in MNIST data set.

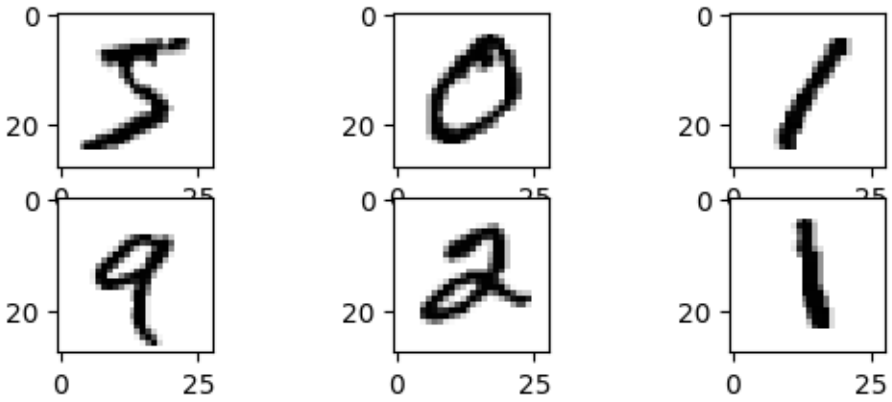


Figure 2. MNIST data set digits example.

Simulated annealing:

Simulated annealing was first introduced by Scott Kirkpatrick in 1982 and was extended and tested as an optimization algorithm in 1983 (Kirkpatrick, Gelatt, & Vecchi, 1983) based on the statistical mechanics of annealing in solids. (Rutenbar, 1989) explain the process and provide an example: "*Consider how to coerce a solid into a low energy state. A low energy state usually means a highly ordered state, such as a crystal lattice; a relevant example here is the need to grow silicon in the form of highly ordered, defect-free crystal for use in semiconductor manufacturing. To accomplish this, the material is annealed: heated to a temperature that permits many atomics rearrangements, then cooled carefully, slowly, until the material freezes into a good crystal*".

The annealing nature is described by the law of thermodynamics that states that at temperature t, the probability of an increase in energy of magnitude ΔE, is given by:

$$P(\Delta E) = \exp\left(-\Delta E / kt\right) \quad (1)$$

Where k is a constant known as Boltzmann's constant.

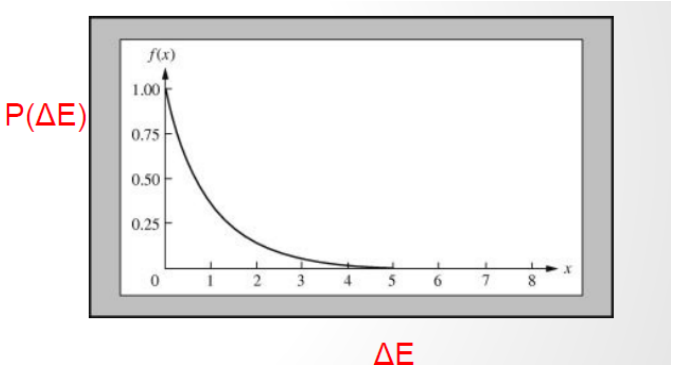


Figure 3. Simulated annealing graph.
(Johnson, Other optimization methods, 2023)

The probability of changing to higher energy decreases with the size of the energy jump. What means that higher energy temperature makes larger energy changes more likely.

Simulated annealing as optimization algorithm equals the energy with fitness what is interpreted as the degree to which particles jump around is controlled by the temperature, then this particles behavior works like a mutation rate (Johnson, Other optimization methods, 2023).

The simulated annealing pseudocode was implemented as below:

1. Mutate by a random amount, small changes are more likely.
    a. Add a small extra probability to change the gene related to kernel to 5%.
    b. Add a small extra probability to change the gene related to gamma to 5%.
2. Measure change in fitness given by $\Delta F = Fnew - Fold$.
    a. If ΔF is greater than 0, the move is accepted.
    b. If ΔF is less than 0 accept the move with a probability $P = \exp\left({}^{-\Delta F}/_{t}\right)$.
    c. End if statement.
3. Reduce the temperature t by small increment Δt.
4. Go to 1.

As represented in the pseudocode, an extra probability was added to the mutation when the gene to modify is the one related to kernel or gamma. Because a change in these specific hyperparameters leads to a big change in fitness and the changes in fitness do not have to be that relevant, also, those changes usually lead to significantly lower fitness.

Finally, to obtain a characteristic result, the algorithm was executed 10 times.

Genetic algorithm:

The genetic algorithm is based on Charles Darwin's natural selection theory. Darwin says "*if variations useful to any organic being do occur, assuredly individuals thus characterized will have the best chance of being preserved in the struggle for life; and from the strong principle of inheritance, they will tend to produce offspring similarly characterized. This principle of preservation, I have called, for the sake of brevity, Natural Selection*" (Darwin, 2006).

The fundamentals of a genetic algorithm according to (Thede, 2004) are **population**, **individual**, **fitness** and **selection**.

- **Population:** Collection of candidate solutions that are considered during the course of the algorithm. Over the generations of the algorithm, new members are "*born*" into the population, while others "*die*" out of the population.
- **Individual:** Single solution in the population.
- **Fitness:** Measure of how "*good*" the solution is represented by the individual. Usually, the higher the fitness, the better - Depending on the problem to be solved.
- **Selection:** The survival of the fittest individual. Individuals that are selected for "*breeding*" – That is where the **crossover** takes place, based on the parent's

fitness values where a high fitness makes more likely an individual to reproduce. Moreover, during each generation, there is a small probability for each individual to **mutate**, which changes the individual in some small way.

The genetic algorithm pseudocode implemented based on (Thede, 2004) and (Johnson, Evolution as algorithm, 2023) is shown below:

1. Create a **population** of random candidate solutions, number of genes, and number of generations named *pop, NoGenes, NoIndividuals* and *NoGenerations*.
2. Calculate **Fitness** for the initial population.
3. Until the algorithm termination conditions are met, do the following (each iteration is a generation):
    a. Create an empty population named *newPop.*
    b. While *newPop* is not full:
        i. **Select** parents proportional to **fitness**.
            1. Parents selected by rank selection.
        ii. **Crossover** parents to create children.
        iii. **Mutate** children.
        iv. Calculate fitness of the children.
        v. **Add** children to *newPop.*
    c. Replace *pop* with *newPop.*
4. Select the individual from new*pop* with the highest fitness as the solution to the problem.

Same as in SA algorithm, an extra probability was added to the mutation. If the probability of mutation is met, then, a new probability must be met to change the kernel or gamma parameters.

Finally, to obtain a characteristic result, the algorithm was executed 10 times as the simulated annealing algorithm. This is due to the stochasticity of the algorithms, therefore, is necessary to test the model multiple times.

Microbial genetic algorithm:

The microbial genetic algorithm is based on the behavior of microbial reproduction; therefore, it does not exist the population-based evolution. Where the daughter inherits genes from the parents, then, mutate and the fitness of the offspring is calculated and added to the new population. The previous method has a higher complexity and requires more computer resources and is too demanding to be implemented. (Harvey, 1996) mentions that the microbial genetic algorithm is a way to simplify a genetic algorithm in which forms retain selection, recombination, and mutation.

To provide an overview of the structure of a microbial genetic algorithm, figure 4 shows how the algorithm transforms the population. The first important factor is that the selection of the 2 individuals is made stochastically, then is compared the fitness of each individual,

the individual with the highest fitness passes its genes to the loser and the loser is mutated depending on the probability of mutation. Finally, both individuals are placed back in the population.

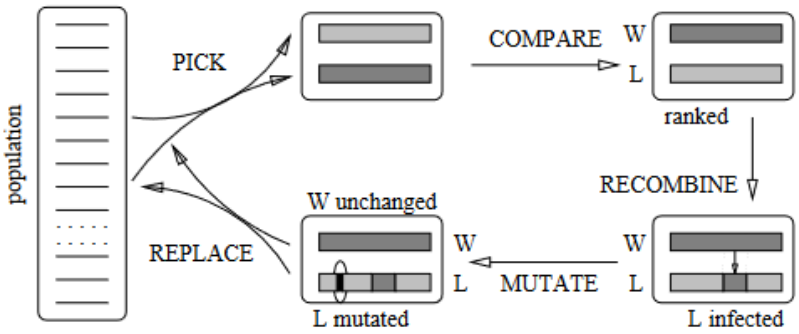


Figure 4. Microbial genetic algorithm diagram

(Harvey, 1996)

The implementation of the microbial genetic algorithm is detailed in the following pseudocode:

1. Initialize the number of genes and generations.
2. Initialize the population.
3. Calculate population fitness.
4. For i in the length of the generations:
    a. Select randomly two individuals from the population.
    b. Apply tournament selection and select the individual with better fitness.
        i. Compare the fitness of individual 1 and 2.
        ii. Decide the winner which has the higher fitness.
        iii. Set the crossover and mutation probability.
    c. For j in the length of the population:
        i. Copy the genes of the winner to loser with probability of crossover(pc).
        ii. Mutate the genes of the loser with probability of mutation(pm).
    d. End
    e. Update the loser's new fitness value in the population.
5. End

Some important key points to simulate adequately the microbial genetic algorithm used:

- The initial population is initialized randomly.
- The final fitness is stored in an array.
- Fitness is calculated first for the initial population and after crossover and mutation, fitness is calculated once again.
- The output is an array with the fitness of each solution.

- The algorithm is executed 10 times to obtain characteristic results.

Fitness function:

The fitness function used was the F1 score of the SVM classifier. The SVM was trained with random values in the training subset. Furthermore, F1 score was calculated using the *sklearn* Python library.

3 Results:

Simulated annealing results:

Simulated annealing starts with a random possible solution, that in this case is a set of SVM classifier hyperparameters. Then, mutate the hyperparameters given a small change, if new fitness is higher, the change is always accepted. However, if the change is negative the change is accepted under a probability given by the difference in fitness and the time.

Due to the stochasticity of the algorithm, is recommended to execute the code several times, thus, obtain characteristic results and be able to make conclusions under enough data.

As shown in figure 5, a simulated annealing algorithm found a set of hyperparameters that provide an F1 score value of 0.957.

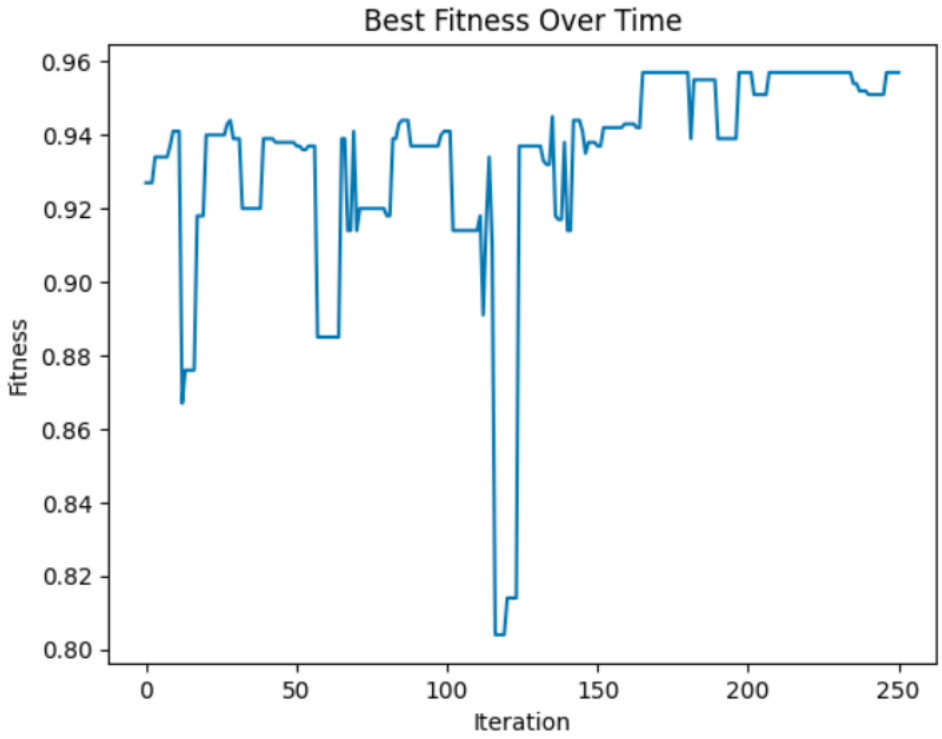


Figure 5. Simulated Annealing - F1 Score - 0.957

Furthermore, in figure 6 the algorithm evidences the ability to start with a set of hyperparameters that evidence a low F1 score performance, however, throughout iterations find better solutions in the solution space and end with a high performance.

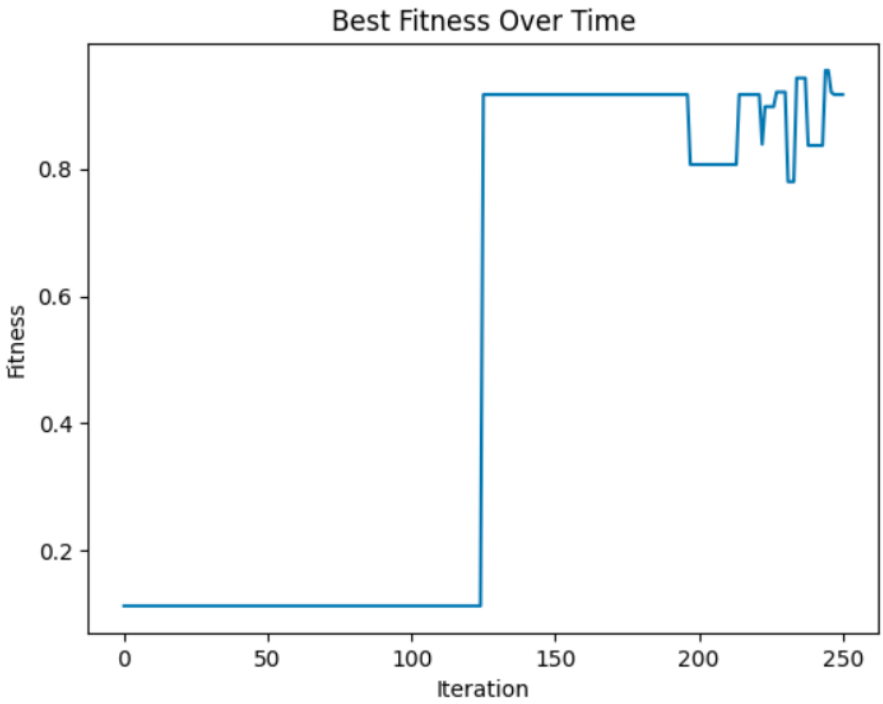


Figure 6. Simulated Annealing - F1 Score - 0.917

On the other hand, is possible to achieve lower fitness. In figure 7, is shown that the random initial hyperparameters had a good performance. Nonetheless, this can be fixed by increasing the number of iterations.

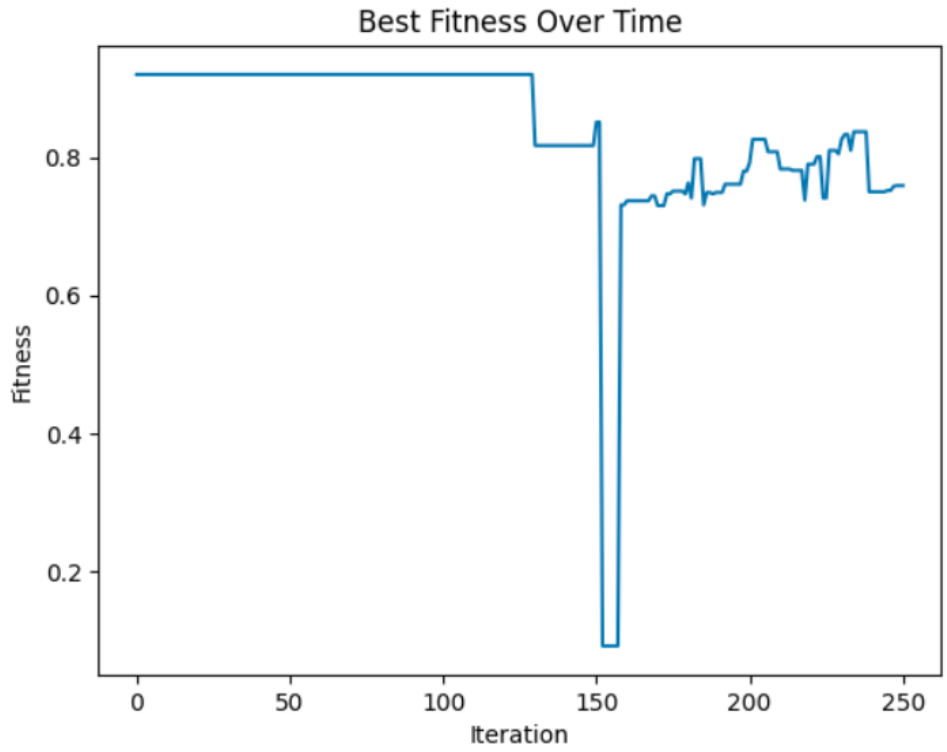


Figure 7. Simulated Annealing - F1 score - 0.759

The best three set of hyperparameters throughout the experimentation were:

Rank\Parameters	Kernel	C	Degree	Gamma	Coef0	F1 Score
1.	rbf	10.5	6	scale	0.9	0.974
2.	rbf	3.0	9	scale	0.25	0.958
3.	poly	10.0	2	auto	0.9	0.937

Table 1. Top 3 simulated annealing hyperparameters set

On the other hand, the average fitness in 10 executions was 0.819. This us given by a run that had low fitness result. That solution fitness was 0.104. Because the hyperparameters set had a considerable change at the beginning, as before that change the fitness was around 0.85. However, the algorithm could not find a better solution in 250 iterations.

The hyperparameters set that had that low fitness was Kernel = ‘rbf’, C = 10, Degree = 1, Gamma = ‘auto’ and Coef0 = 0.15.

Genetic algorithm results:

The genetic algorithm starts with a random population that is made of hyperparameters sets or possible solutions. Then, two individuals are picked using rank selection to produce and offspring and mutate (the offspring) given the mutation probability. The new solutions are used to create a new population and the individual (solution) with the higher fitness is selected as the result.

Same as simulated annealing, is recommended to execute the code several times, thus, obtaining characteristic results and be able to make conclusions under enough data.

The genetic algorithm performed very well in finding the best solution. As mentioned before, the selection of the parents was made by rank selection. Hence, there is a high possibility to always crossover the best individuals in the population. In figure 8 is evidenced that after 250 generations the individual with lowest fitness was 0.94 and by the end the best fitness was increased to 0.96.

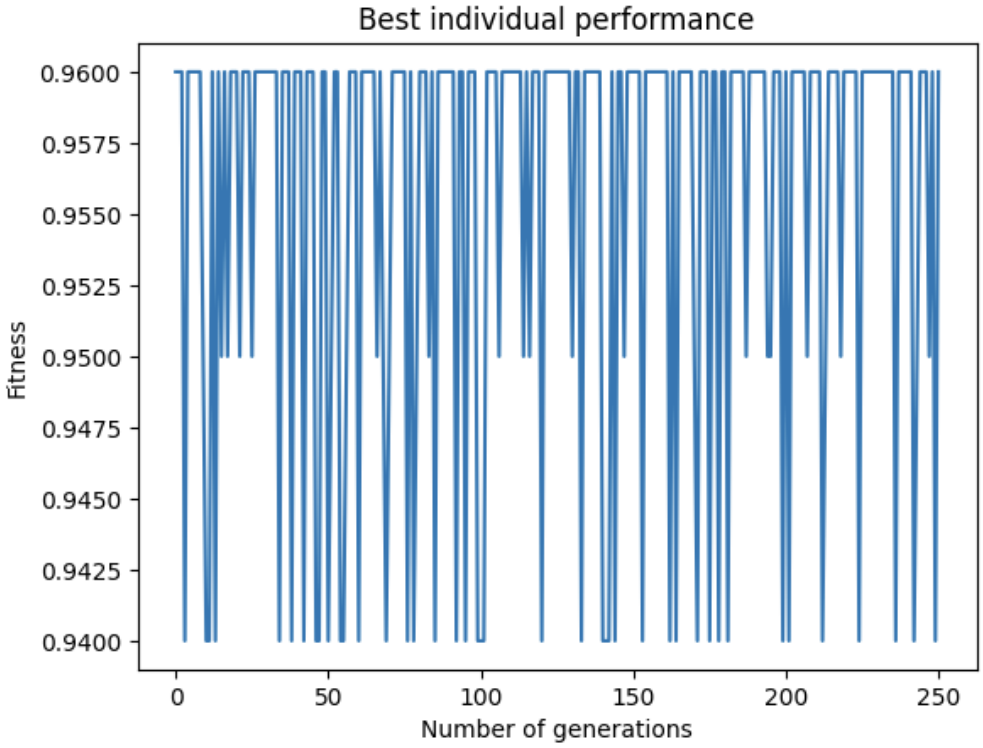


Figure 8. Genetic Algorithm - F1 Score - 0.96

In addition, the best solution is always a high F1 score value. However, took more time than simulated annealing to find the hyperparameters. Figures 9, 10, 11 and 12, evidence the fitness through generations.

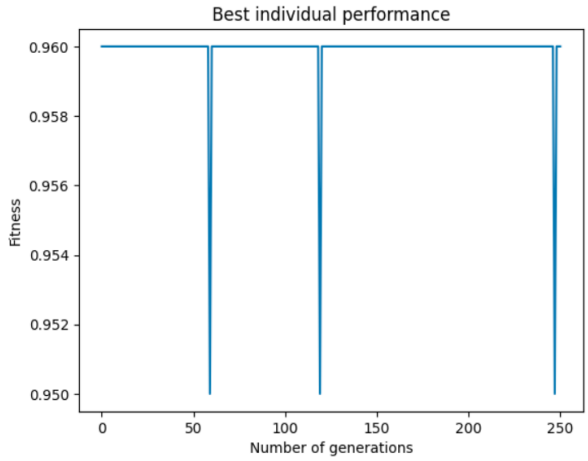


Figure 9. Genetic algorithm result (a)

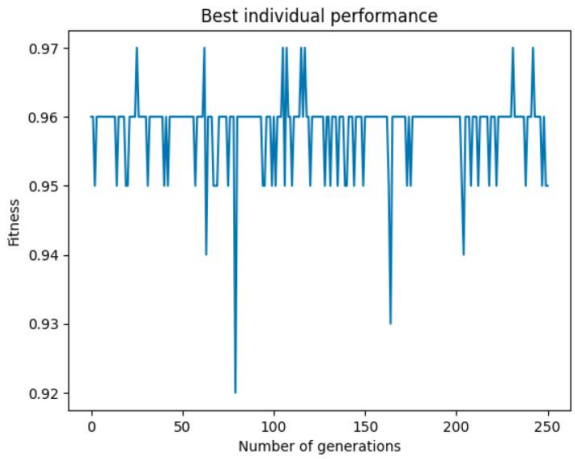


Figure 10. Genetic algorithm result (b)

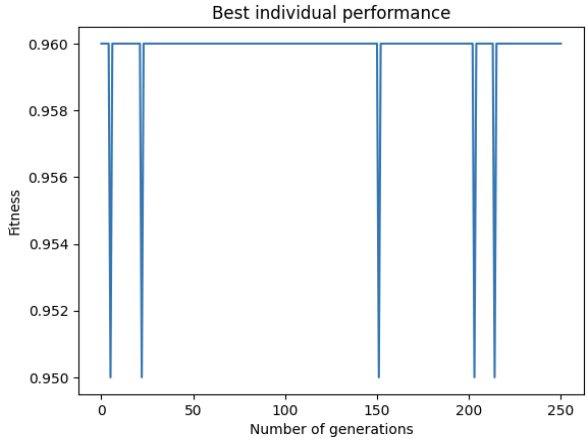


Figure 11. Genetic algorithm result (c)

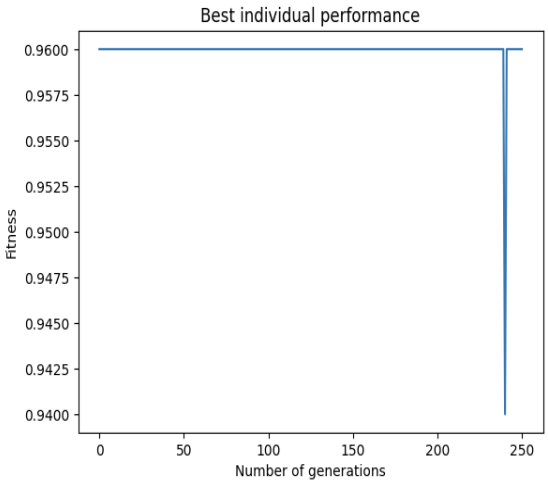


Figure 12. Genetic algorithm result (d)

Additionally, the best three set of hyperparameters throughout the experimentation were:

Rank\Parameters	Kernel	C	Degree	Gamma	Coef0	F1 Score
1.	rbf	4	5	scale	0.95	0.96
2.	poly	14	4	scale	0.8	0.96
3.	poly	1.5	4	scale	0.8	0.95

Table 2. Top three Genetic algorithm hyperparameters set

Similarly, the average fitness was 0.957 during the 10 runs. Also, the lowest fitness evidenced was 0.95 with different hyperparameters combinations such as: Kernel = ‘poly’, C = 1.5, Degree = 4, Gamma = ‘scale’ and Coef0 = 0.8.

Microbial GA:

The microbial GA is similar to the standard GA. An initial population is created, and two random individuals are picked. Nevertheless, in this case the individuals picked are randomly selected with the same probability. Then, the fitness of both solutions is estimated and compared. The loser genes are replaced by the winner genes and have a mutation probability. Finally, as in the standard GA, the individual with the highest fitness score is picked as the solution.

As done for the previous algorithms, the model was executed multiple times, the best fitness score is taken from the population fitness list and the hyperparameters as done with the standard GA is taken as the best solution. Figure 13 shows the evolution of the hyperparameters throughout 250 iterations.

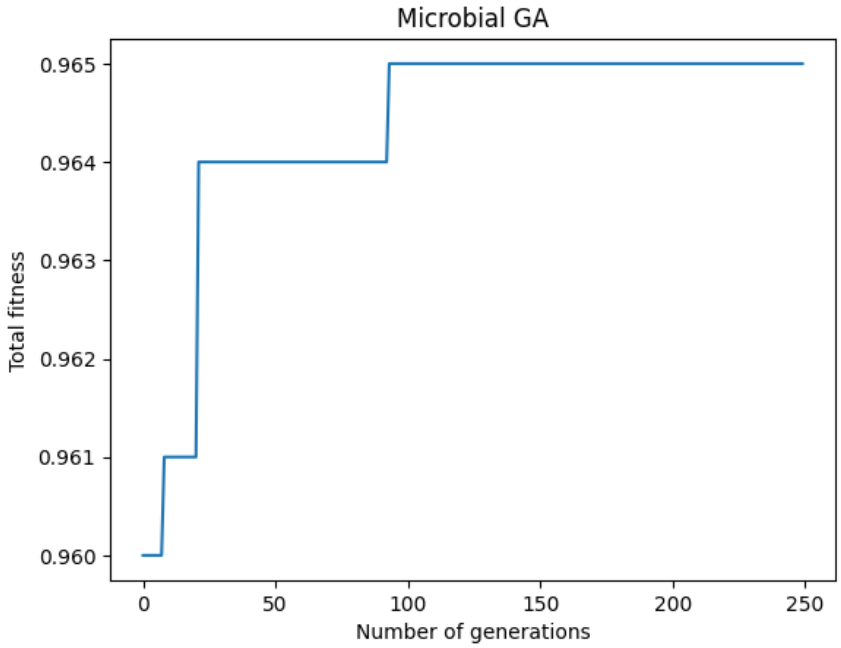


Figure 13. Genetic Algorithm - F1 Score - 0.965

Moreover, after multiple executions, as mentioned above, the change in fitness is minimal. Figures 14, 15 and 16 show a maximum change of 0.005.

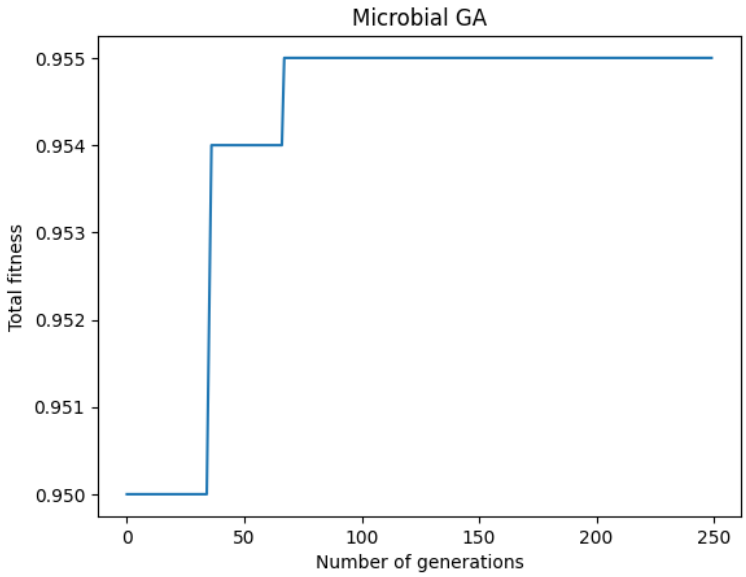


Figure 14. Genetic Algorithm - F1 Score - 0.955

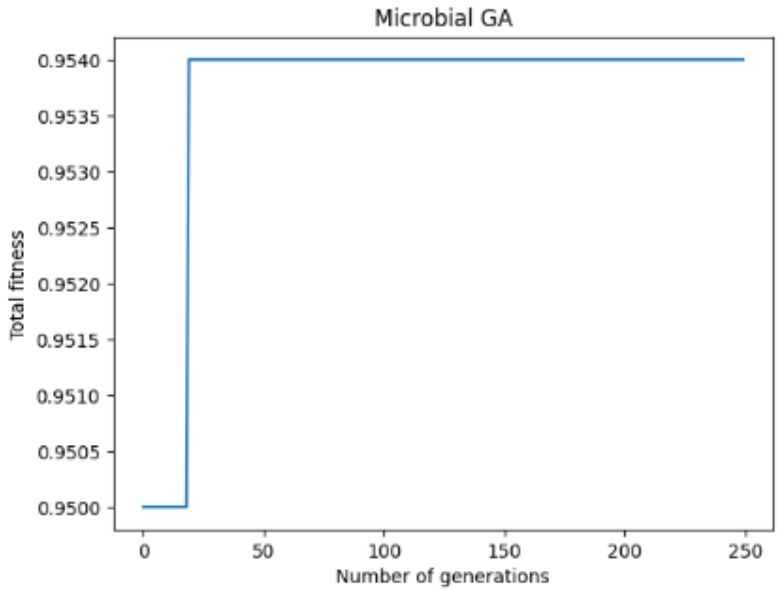


Figure 15. Genetic Algorithm - F1 Score - 0.954

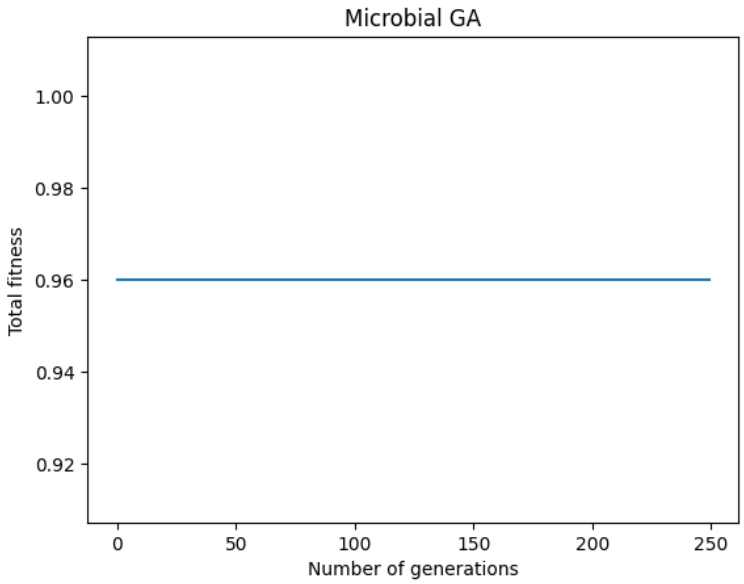


Figure 16. Genetic algorithm - F1 score - 0.96

Furthermore, the best three set of hyperparameters throughout the experimentation were:

Rank\Parameters	Kernel	C	Degree	Gamma	Coef0	F1 Score
1.	rbf	5.5	5	scale	0.95	0.97
2.	rbf	4	4	scale	0.05	0.965
3.	poly	1.5	4	scale	0.35	0.96

Table 3. Top three Microbial Genetic algorithm hyperparameters set

Likewise, the average fitness using the microbial GA is 0.962. As mentioned above, this algorithm does not offer a considerable change in fitness. Nonetheless, its computational complexity is not high, hence, multiple runs does not take a lot of time. Moreover, the lowest fitness was 0.954 with the following hyperparameters set: Kernel = ‘rbf’, C = 11.5, Degree = 1, Gamma = ‘scale’ and Coef0 = 0.75.

4 Discussion:

The time and effort invested in hyperparameters tuning of machine learning algorithms evidence the complex operation that it is. That is why the main goal of the experiment was to compare three different optimization algorithms to find the best hyperparameters for a SVM classifier using MNIST dataset. The algorithms proposed were Simulated Annealing, Genetic Algorithm and Microbial Genetic Algorithm. The evaluation process was focused on the evaluation metric F1 score where the best fitness and hyperparameters sets were stored. The hyperparameters used were Kernel, C, Degree, Gamma and Coef0.

The results of the experiment were balanced, no algorithm stands out to be the clear best option to optimize a SVM classifier. Nonetheless, the algorithm that achieved the highest fitness scores was SA. Since simulates annealing accepts more changes in the solution in the first iterations, that allows the possibility of try multiple combinations of hyperparameters. The results evidence that SA algorithm optimizes fitness considerably. Even if the initial randomly generated solution has a low fitness, the algorithm changes enough the solution throughout the iterations that allows exploring the solution space better.

The standard GA results are similar to the SA algorithm. However, is a more complex algorithm, and the time per iteration is higher. Nonetheless, the results evidence that using the standard GA is likely to have more than 0.9 in F1 score metric. On the other hand, microbial GA does not change the best hyperparameters enough to increase fitness considerably. Therefore, the genotypes do not change enough to explore widely the solution space. However, among the random created population, there is a high probability that the randomly generated initial population have various individuals with high performances.

To reproduce this experiment in other classification models, is necessary to understand the nature of the algorithms, as the proposed optimization algorithms work under the premise that the mutation does not have to be big enough to change the genes too much, just enough to represent a change and make permit the algorithm make the decisions. Thus, the extra probabilities in changing the kernel and gamma were applied, as a change in these parameters lead to a big change in fitness.

5 Future work:

The computational complexity of using the Genetic Algorithm is higher than using the microbial GA or the SA algorithm. Hence, in the future the data structures used, and algorithm processes can be checked to improve the execution time and make the standard GA more viable. Furthermore, a suggested way to improve the results is to increase the population in standard and microbial GA. Also, increasing the mutation rate in both to no more than 15% may help to obtain better results. On the other hand, as mentioned before, the algorithm with the best performance and less computational complexity was the SAA algorithm. Nevertheless, is possible to improve its performance by increasing the maximum iterations and changing by the small factor the cooling rate. This helps the algorithm to explore lower fitness and avoid getting stuck in suboptimal solutions.

6 References:

Ali, Y. A., Awwad, E. M., Al-Razgan, M., & Maarouf, A. (2023). *Hyperparameter Search for Machine Learning Algorithms for Optimizing the Computational Complexity.* Riyadh: Processes. Retrieved from https://www.mdpi.com/2227-9717/11/2/349

Alibramhim, H., & Ludwig, S. A. (2021). *Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization.* Krakow: 2021 IEEE Congress on Evolutionary Computation.

Aszemi, N. M., & Dominic, P. (2019). *Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms.* Seri Iskandar: International Journal of Advanced Computer Science and Applications. Retrieved from https://thesai.org/Downloads/Volume10No6/Paper_38-Hyperparameter_Optimization_in_Convolutional_Neural_Network.pdf

Darwin, C. (2006). *On the origin of species in from so simple a beginning: the four great books of Charles Darwin. edited, with introductions, by Edward O. Wilson. W. W. Norton & Company.* New York.

Elgeldawi, E., Sayed, A., Galal, A. R., & Zaki, A. M. (2021). *Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis.* Informatics. Retrieved from https://www.mdpi.com/2227-9709/8/4/79

Harvey, I. (1996). *The microbial genetic algorithm.* Brighton: University of Sussex.

Johnson, C. (2023). *Evolution as algorithm.* Brighton: University of Sussex.

Johnson, C. (2023). *Other optimization methods.* University of Sussex. Brighton: University of Sussex.

Kennedy, J., & Eberhart, R. (1995). *Particle swarm optimization.* Purdue school of engineering and technology. Retrieved from https://ieeexplore-ieee-org.sussex.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=488968

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). *Optimization by simulated annealing* (Vol. 220). Science. Retrieved from https://www.science.org/doi/epdf/10.1126/science.220.4598.671

LeCun, Y., Cortes, C., & Burges, C. (n.d.). *The MNIST database of handwritten digits.* Retrieved from http://yann.lecun.com/exdb/mnist/

Liang, J.-Z. (2004). *SVM multi-classifier and Web document classification.* Shanghai: IEEE.

Purba, K. R., Asirvatham , D., & Murugesan, R. K. (2020). *Classification of instagram fake users using supervised machine learning algorithms.* Subang Jaya: International Journal of Electrical and Computer Engineering. Retrieved from

https://pdfs.semanticscholar.org/14f1/b5ab561f3a26d48ca2a46fcebe4c172e37b3.pdf

Rutenbar, R. A. (1989). *Simulated annealing algorithms: An overview.* IEEE Circuits and Devices Magazine. Retrieved from https://ieeexplore-ieee-org.sussex.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=17235

Tayebi, M., & El Kafhali, S. (2021). *Hyperparameter Optimization Using Genetic Algorithms to Detect Frauds Transactions.* Springer: Proceedings of the International Conference on Artificial Intelligence and Computer Vision. Retrieved from https://link.springer.com/chapter/10.1007/978-3-030-76346-6_27#chapter-info

Thede, S. M. (2004). *An introduction to genetic algorithms.* Greencastle: DePauw University. Retrieved from https://www.researchgate.net/profile/Scott-Thede/publication/228609251_An_introduction_to_genetic_algorithms/links/00b7d52cdb71f1e8bf000000/An-introduction-to-genetic-algorithms.pdf

Wang, D., Tan, D., & Liu, L. (2017). *Particle swarm optimization algorithm: an overview.* Soft comput. Retrieved from https://link.springer.com/article/10.1007/s00500-016-2474-6#Abs1

Wassan, S., Chen, X., Shen, T., Waqar, M., & Jhanjhi, N. (2021). *Amazon product sentiment analysis using machine learning techniques.* Revista Argentina de Clinica Psicologica. Retrieved from https://www.researchgate.net/profile/Sobia-Wassan-2/publication/349772322_Amazon_Product_Sentiment_Analysis_using_Machine_Learning_Techniques/links/60411e09a6fdcc9c78121992/Amazon-Product-Sentiment-Analysis-using-Machine-Learning-Techniques.pdf

Appendices

The source code used in this experiment can be found here: [GitHub repository](#).

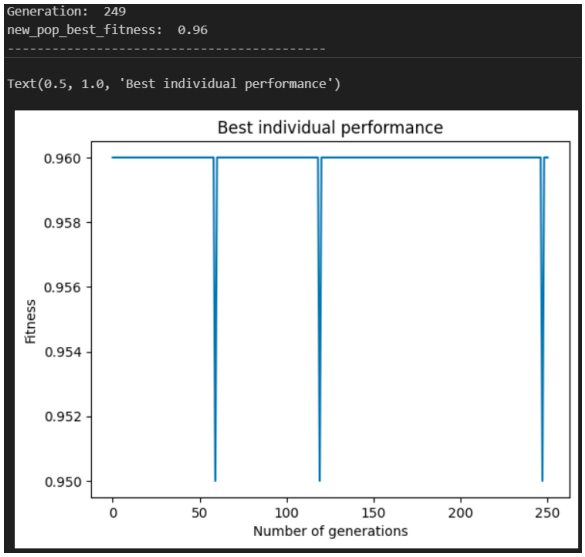
Appendix 1:

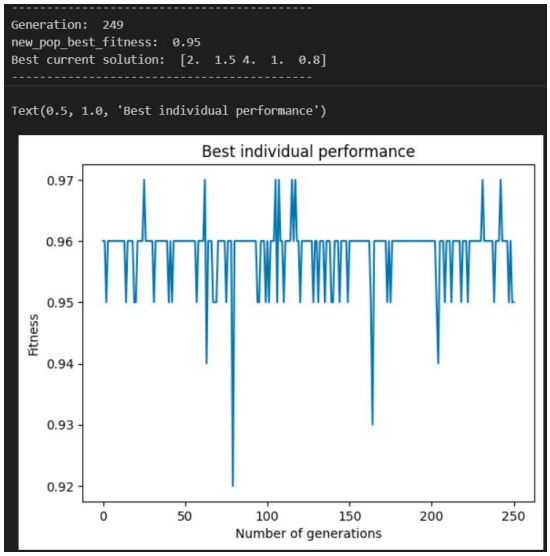
Results for the multiple executions using the standard GA algorithm. The kernel and gamma were encoded to be mutated throughout the experiment. The values for kernel are:

1. Kernel = 1 = ‘linear’
2. Kernel = 2 = ‘poly’
3. Kernel = 3 = ‘rbf’
4. Kernel = 4 = ‘sigmoid’

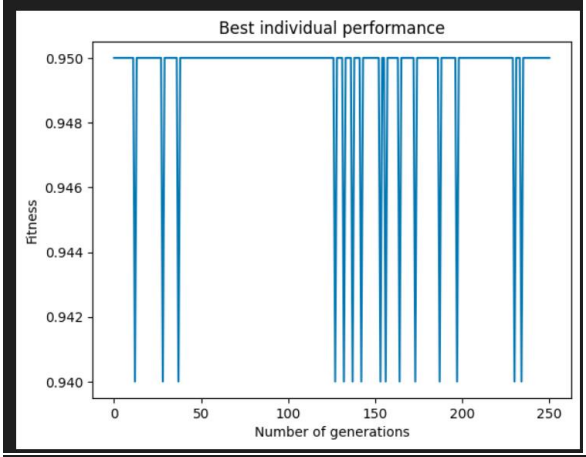
The values for gamma are:

1. Gamma = 1 = ‘scale’
2. Gamma = 2 = ‘auto’

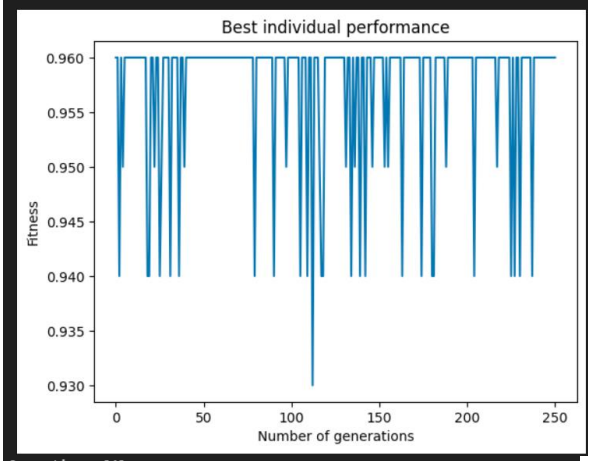




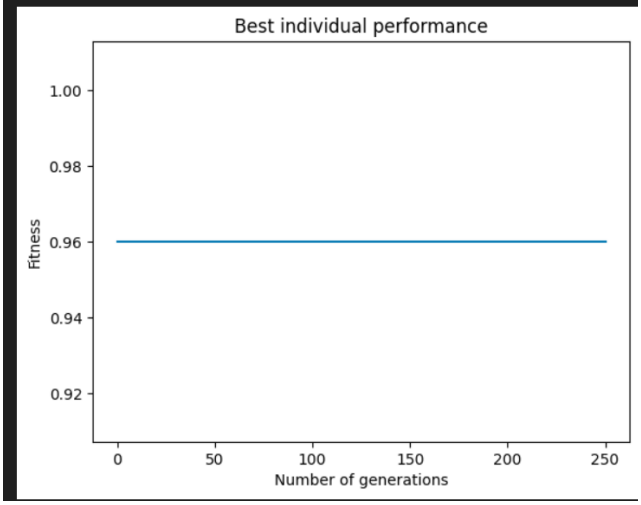
```
Generation:  249
new_pop_best_fitness:  0.95
Best current solution:  [3.   5.5  5.   1.   0.35]
-----------------------------------------

Text(0.5, 1.0, 'Best individual performance')
```



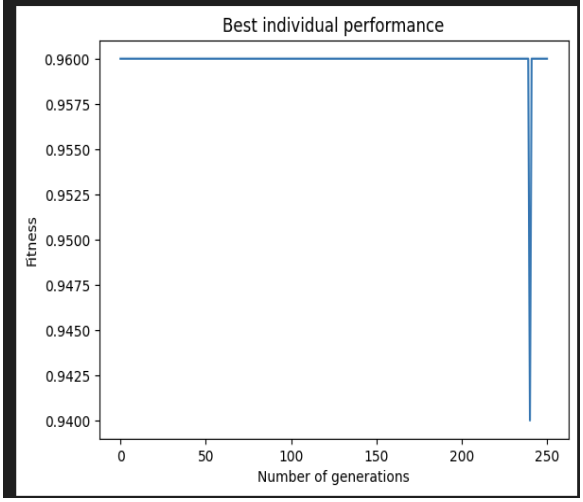

```
Generation:  249
new_pop_best_fitness:  0.96
Best current solution:  [3.   5.5  5.   1.   0.85]
-----------------------------------------

Text(0.5, 1.0, 'Best individual performance')
```



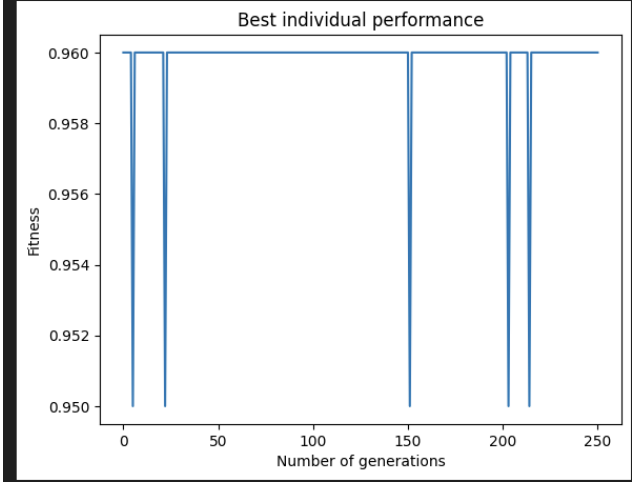

```
Generation:  249
new_pop_best_fitness:  0.96
Best current solution:  [3.  5.5 4.  1.  0.8]
-----------------------------------------

Text(0.5, 1.0, 'Best individual performance')
```



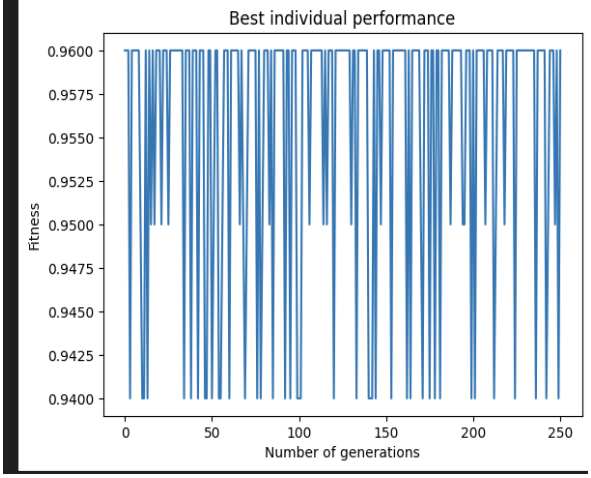

```
Generation:  249
new_pop_best_fitness:  0.96
Best current solution:  [3.  3.5 5.  1.  0.8]
-----------------------------------------

Text(0.5, 1.0, 'Best individual performance')
```

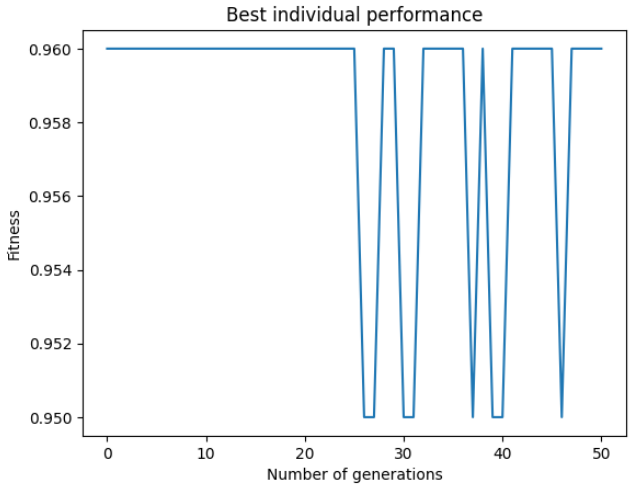


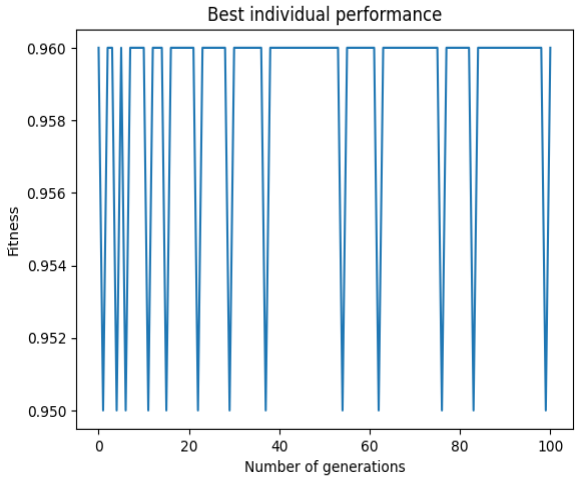
```
Generation:  249
new_pop_best_fitness:  0.96
Best current solution:  [ 2.  14.   4.   1.   0.8]
-----------------------------------------

Text(0.5, 1.0, 'Best individual performance')
```




```
Generation:  249
new_pop_best_fitness:  0.96
Best current solution:  [3.   4.   5.   1.   0.95]
-----------------------------------------

Text(0.5, 1.0, 'Best individual performance')
```



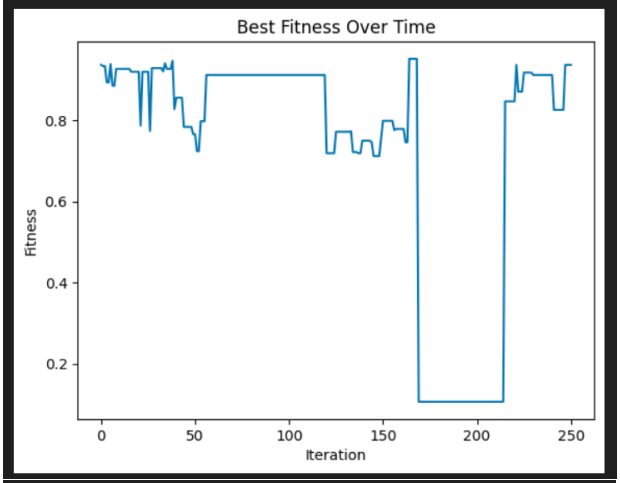




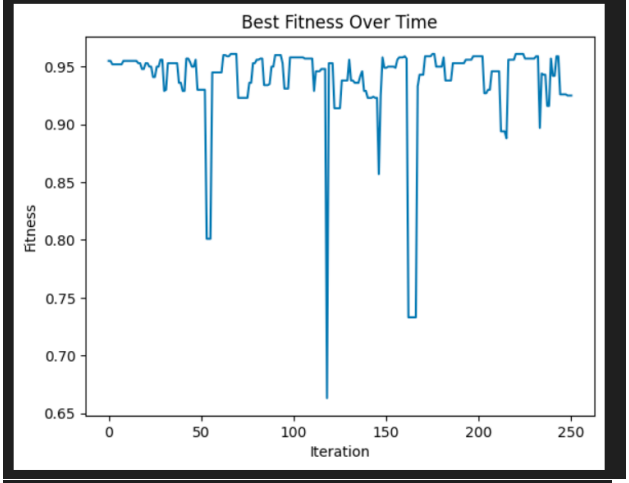
Appendix 2:

Results for the multiple executions using the SA algorithm.

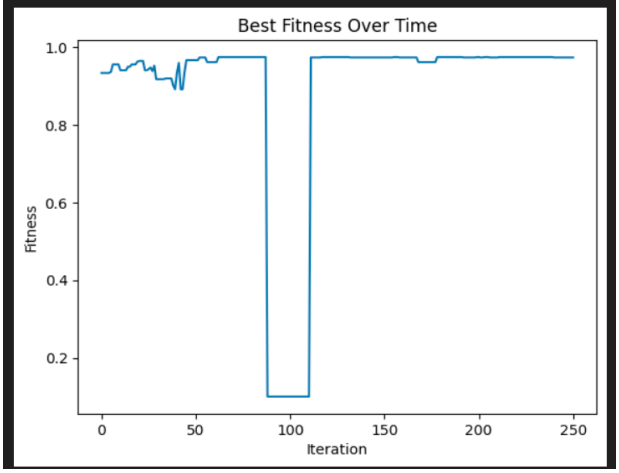
```
new_fitness: 0.937
change_in_fitness: 0.0
Current solution: ['poly', 10.0, 2, 'auto', 0.9]
fitness_over_time: [0.937, 0.934, 0.934, 0.894, 0.894, 0.939, 0.886, 0.885,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 93
iteration: 249
----------------------------------------------------
Best solution: ['poly', 10.0, 2, 'auto', 0.9]
```



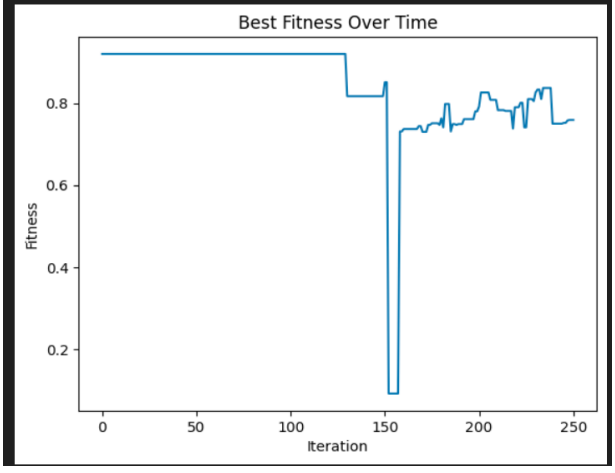

```
new_fitness: 0.925
change_in_fitness: 0.0
Current solution: ['poly', 13.5, 1, 'scale', 0.7000000000000001]
fitness_over_time: [0.955, 0.955, 0.952, 0.952, 0.952, 0.952, 0.952, 0.952, 0.9
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 93.26
iteration: 249
----------------------------------------------------
Best solution: ['poly', 13.5, 1, 'scale', 0.7000000000000001]
```



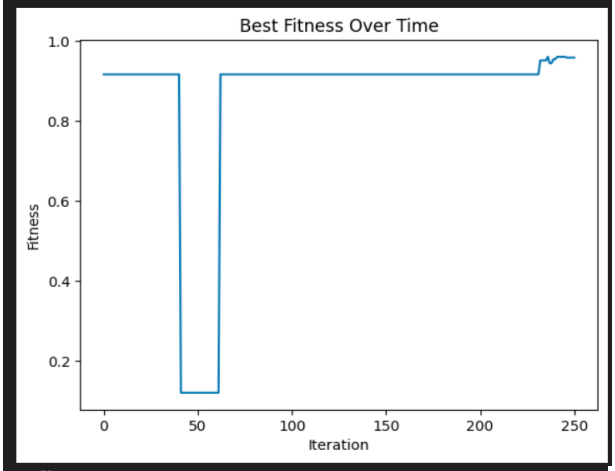

```
new_fitness: 0.974
change_in_fitness: 0.0
Current solution: ['rbf', 10.5, 6, 'scale', 0.9]
fitness_over_time: [0.934, 0.934, 0.934, 0.934, 0.934, 0.937, 0.956, 0.956,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 93
iteration: 249
----------------------------------------------------
Best solution: ['rbf', 10.5, 6, 'scale', 0.9]
```



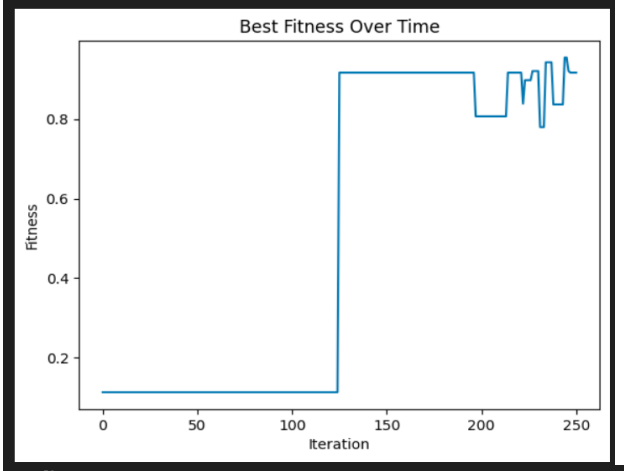

```
new_fitness: 0.7590000000000001
change_in_fitness: 0.0
Current solution: ['sigmoid', 7.0, 1, 'scale', 0.5]
fitness_over_time: [0.92, 0.92, 0.92, 0.92, 0.92, 0.92, 0.92, 0.92, 0.92, 0
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 93
iteration: 249
----------------------------------------------------
Best solution: ['sigmoid', 7.0, 1, 'scale', 0.5]
```



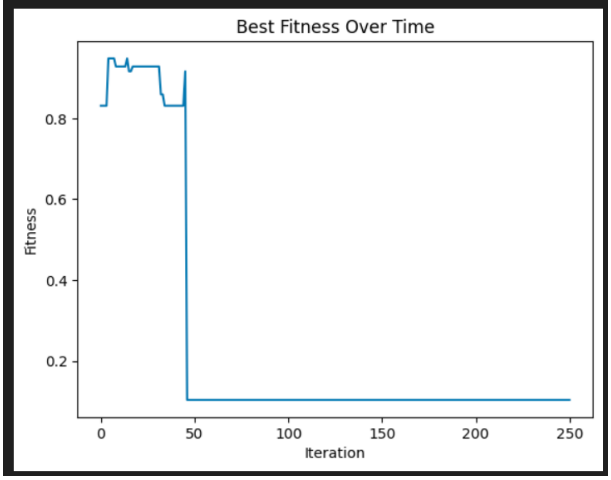
```
new_fitness: 0.958
change_in_fitness: 0.0
Current solution: ['rbf', 3.0, 9, 'scale', 0.25]
fitness_over_time: [0.916, 0.916, 0.916, 0.916, 0.916, 0.916, 0.916, 0.916,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 9
iteration: 249
----------------------------------------------------
Best solution: ['rbf', 3.0, 9, 'scale', 0.25]
```



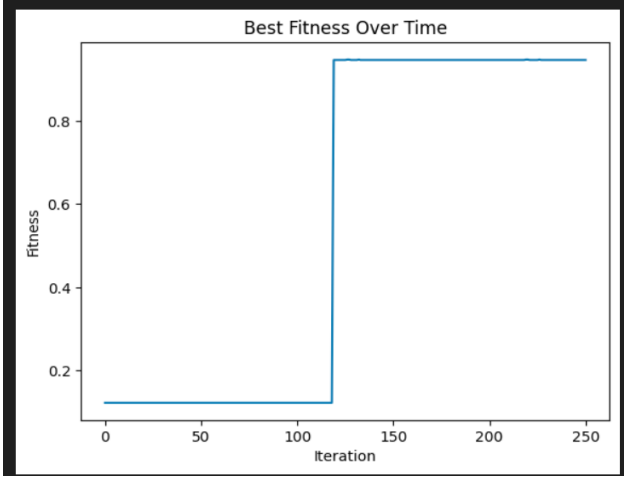

```
new_fitness: 0.917
change_in_fitness: 0.0
Current solution: ['poly', 4.0, 1, 'auto', 0.5]
fitness_over_time: [0.113, 0.113, 0.113, 0.113, 0.113, 0.113, 0.113, 0.113,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 9
iteration: 249
----------------------------------------------------
Best solution: ['poly', 4.0, 1, 'auto', 0.5]
```



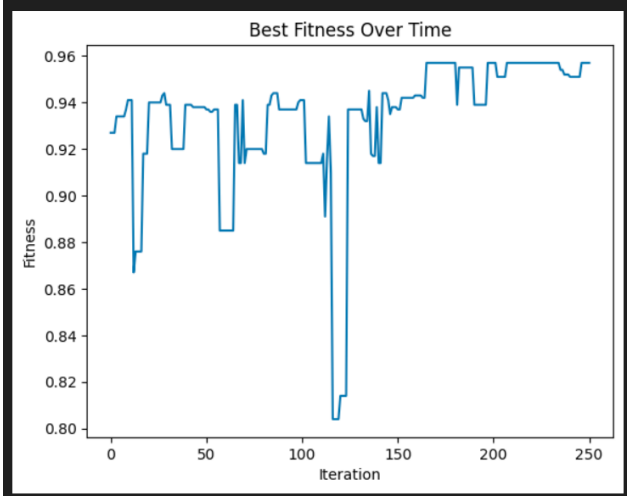

```
new_fitness: 0.104
change_in_fitness: 0.0
Current solution: ['rbf', 10.0, 1, 'auto', 0.15000000000000002]
fitness_over_time: [0.831, 0.831, 0.831, 0.831, 0.948, 0.948, 0.948, 0.948,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 9
iteration: 249
----------------------------------------------------
Best solution: ['rbf', 10.0, 1, 'auto', 0.15000000000000002]
```



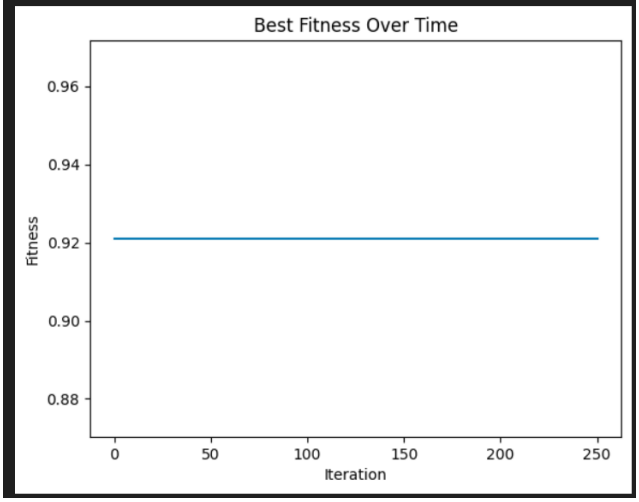

```
new_fitness: 0.9469999999999998
change_in_fitness: 0.0
Current solution: ['rbf', 10.5, 7, 'scale', 0.55]
fitness_over_time: [0.122, 0.122, 0.122, 0.122, 0.122, 0.122, 0.122, 0.122,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 9
iteration: 249
----------------------------------------------------
Best solution: ['rbf', 10.5, 7, 'scale', 0.55]
```



```
new_fitness: 0.957
change_in_fitness: 0.0
Current solution: ['rbf', 8.0, 8, 'scale', 0.55]
fitness_over_time: [0.927, 0.927, 0.927, 0.934, 0.934, 0.934, 0.934, 0.934,
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 93
iteration: 249
----------------------------------------------------------
Best solution: ['rbf', 8.0, 8, 'scale', 0.55]
```

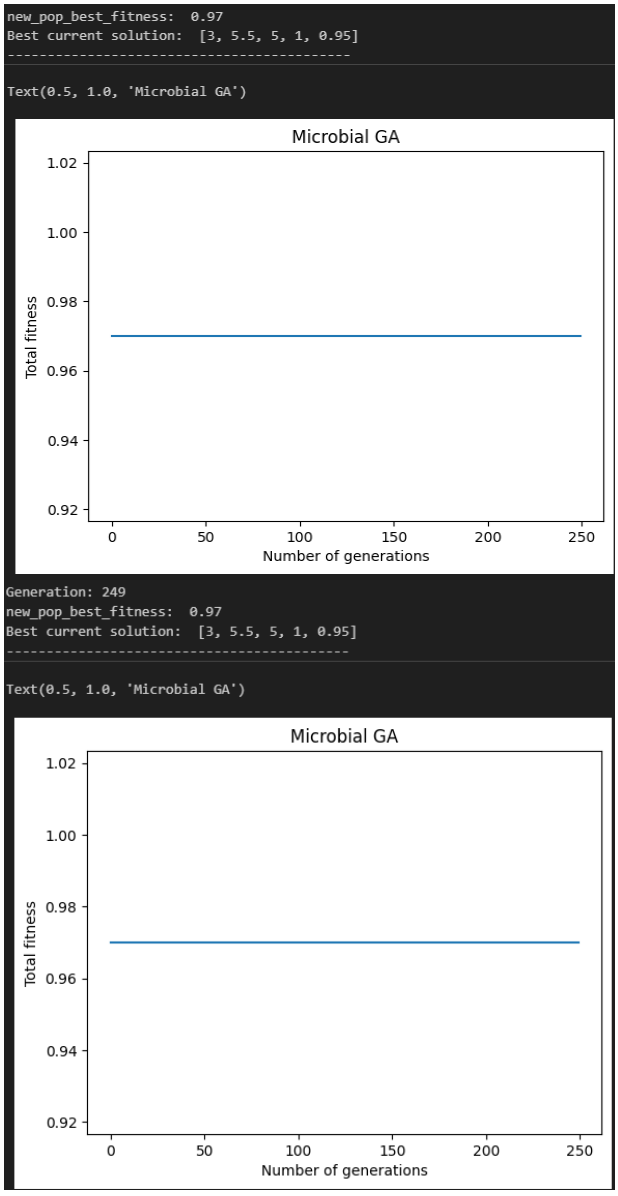


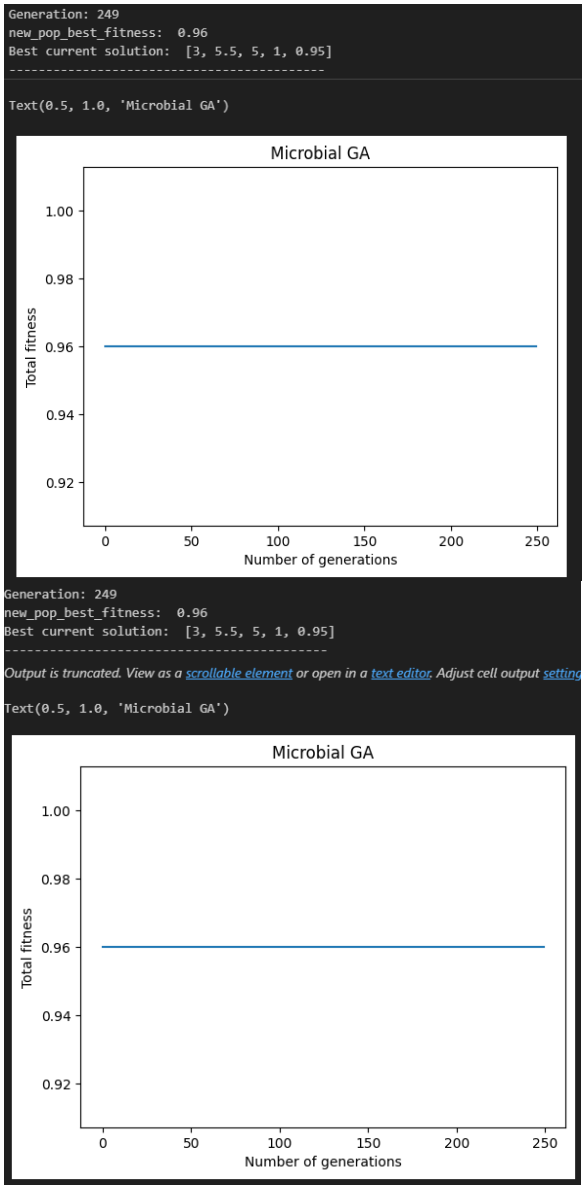

```
new_fitness: 0.9209999999999999
change_in_fitness: 0.0
Current solution: ['linear', 12.5, 6, 'scale', 0.9500000000000001]
fitness_over_time: [0.9209999999999999, 0.9209999999999999, 0.920999999999999
temps: [100, 99.0, 98.01, 97.0299, 96.059601, 95.09900499, 94.1480149401, 93
iteration: 249
----------------------------------------------------------
Best solution: ['linear', 12.5, 6, 'scale', 0.9500000000000001]
```



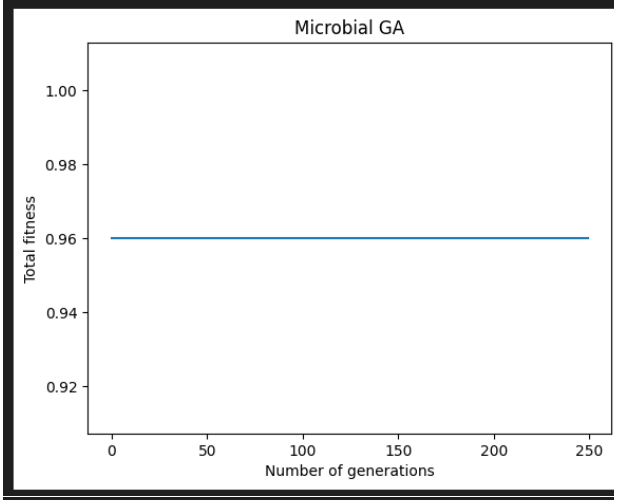
Appendix 3:

Results for the multiple executions using the microbial GA algorithm.

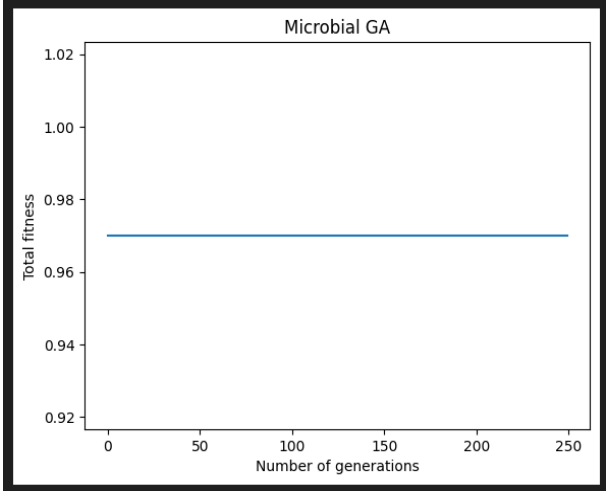




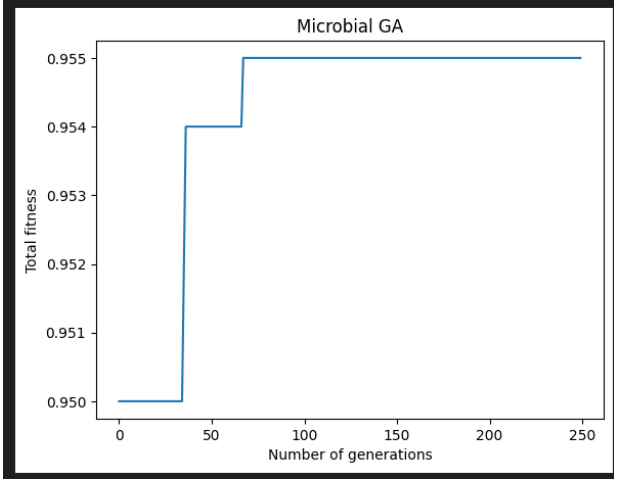
```
Generation: 249
new_pop_best_fitness:  0.96
Best current solution:  [3, 5.5, 5, 1, 0.95]
-----------------------------------------

Text(0.5, 1.0, 'Microbial GA')
```



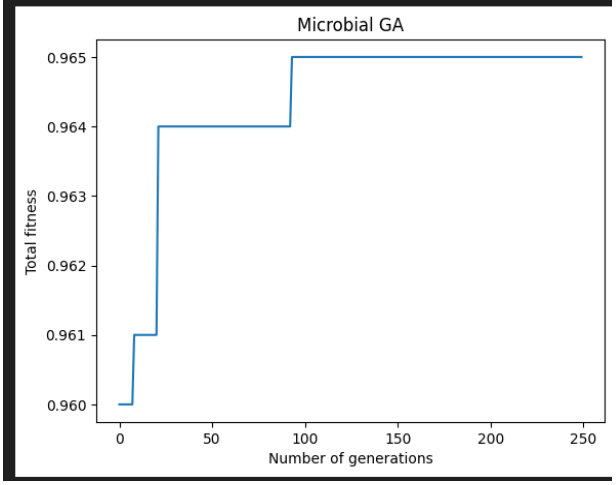

```
Generation: 249
new_pop_best_fitness:  0.97
Best current solution:  [3, 5.5, 5, 1, 0.95]
-----------------------------------------

Text(0.5, 1.0, 'Microbial GA')
```



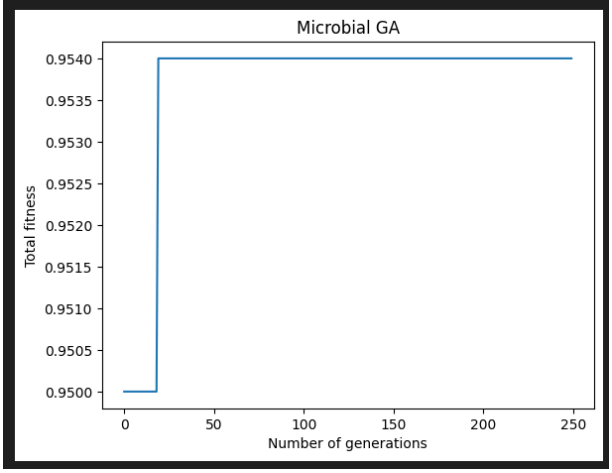

```
Generation: 249
new_pop_best_fitness:  0.955
Best current solution:  [3, 4.5, 9, 1, 0.9]
-----------------------------------------

Text(0.5, 1.0, 'Microbial GA')
```




```
Generation: 249
new_pop_best_fitness:  0.965
Best current solution:  [3, 4.0, 4, 1, 0.05]
-----------------------------------------

Text(0.5, 1.0, 'Microbial GA')
```



```
Generation: 249
new_pop_best_fitness:  0.954
Best current solution:  [3, 11.5, 1, 1, 0.75]
----------------------------------------

Text(0.5, 1.0, 'Microbial GA')
```




```
Generation: 249
new_pop_best_fitness:  0.96
Best current solution:  [3, 5.5, 5, 1, 0.95]
----------------------------------------

Text(0.5, 1.0, 'Microbial GA')
```