TÉCNICO LISBOA

Computational Intelligence for the Internet of Things

Project Report

Heuristic Optimization using GA

Group 5:

Manuel Filipe Martins Passadouro (80840) manuel.passadouro@tecnico.ulisboa.pt
Mariana Lopes Teixeira (96938) mariana.lopes.teixeira@tecnico.ulisboa.pt

Prof. Joao Paulo Carvalho

2023/2024 – 2nd Semester, P4

Contents

1 Introduction

The goal of this project is to develop a solution based on a Genetic Algorithm (GA), that produces the shortest servicing route for an ecopoint system. The routes can vary in length and certain locations have a penalty associated to them (for routes with more than 30 locations), but in essence, this is a variation of the classic Travelling Salesman Problem (TSP).

A TSP can be solved by computing the exact shortest distance but it is an NP-hard problem, meaning that as the number of nodes (ecopoints) grows, the computational requirements increase exponentially. We see can how this increase in complexity make this approach impractical for real-world use cases (assuming one route takes 1μs to compute):

Number of Ecopoints	**Possible Routes**	**CPU Time (s)**
10	$9! = 362{,}880$	0.363
20	$19! = 1.216 \times 10^{17}$	1.216×10^{11}
100	$99! = 9.332622 \times 10^{155}$	9.332622×10^{149}

Table 1: Computation Time for Different Numbers of Ecopoints in the TSP

If we wished to compute the shortest route through all the ecopoints (100), it would take approximately 3×10^{141} years.

Genetic Algorithms are heuristic methods that provide good solutions within a reasonable amount of time. While they do not guarantee the optimal solution, they often find near-optimal solutions quickly, making them suitable for large-scale problems where an exact solution is infeasible.

Genetic Algorithms solve the TSP by emulating the process of natural selection to iteratively evolve a population of potential solutions (routes) in the direction of the optimal route. An initial population of potential routes is generated, where each ecopint to be visited is in a totally randomised order. These routes are then evaluated based on a fitness function, which measures their quality in terms of total distance traveled. The fittest individuals (routes) are selected to reproduce and create offspring through genetic operators of crossover and mutation. This process continues over multiple generations, with each generation producing increasingly better solutions through the principles of survival of the fittest and genetic diversity. Eventually, the algorithm converges towards an optimal or near-optimal solution, which represents the shortest possible tour of the ecopints that need to be visited.

2 Code Structure and Parameters

Using the Distributed Evolutionary Algorithms in Python (DEAP) library, implementation of the solution is fairly straight forward. The process consists essentially in the assembling of a toolbox for the algorithm and defining a fitness function that is suitable for the problem at hand.

The toolbox in DEAP serves as a container for assembling the essential components of an genetic algorithm. It encapsulates how the population and individuals that constitute it are defined as well as the genetic operators for mutation, mating, evaluation and selection. In this case, each individual is defined as a list and thus a population will be a set of lists.

Evaluation of individuals is made based on the fitness function. For this problem, we want the function to return the total distance traveled in a given route, whilst taking into to account the distance penalty of 40% for visiting specific ecopints if more than 30 ecopints have been visited up until that point. In essence, the goal of the algorithm is to minimize this function.

For the mating process, offspring are generated via a crossover operator (*cxOrdered*). This specific crossover function preserves the order of the ecopoints in the parents, which is fundamental for solving TSP-like problems. The mating probability was set to $cxpb = 0.7$.

To boost diversity within a population, we must preform mutation on its individuals. The function *mutShuffleIndexes* performs this mutation by shuffling the genes (that is the ecopoints) the make up each individual. We do not want to set mutation too high so as to avoid instability in the evolution process, the probabilities of mutation can be controlled by parameter *mutpb* (probability of an individual being mutated) and *ind_pb* (probability of each gene being shuffled to another position). The values chosen were $mutpb = 0.2$ and $ind\_pb = 0.05$.

Finally, for the selection mechanism, we used the function *selTournament*. The tournament style selection process is a commonly used in many GA problems, including the TSP. The tournament size was set to $tournsize = 3$.

3 Results

To start testing the code it was decided to start with the parameters population size and number of generations to be 300 and 200 respectively. It was also decided that for the preliminary tests the number of EcoPoints per route would be 30 (as it is the estimated number of EcoPoints in average that the truck will have to pass by):

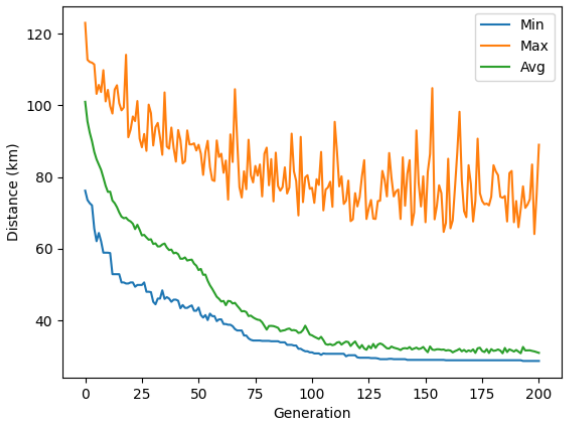


Figure 1: Maximum, Average and Minimum Distace with POP_SIZE = 300 and NUM_GEN = 200 for 30 EcoPoints.

By analyzing the figure above (figure 1), we can conclude that with only 200 generations, the value of the minimum distance does not stabilize. It is necessary to increase both the number of generations and the population size to ensure better exploration of the solution space, prevent premature convergence, and achieve more robust optimization results. With that in mind we tested several more times with different parameters values and starting to increase the number of EcoPoints until we reached the 99 EcoPoints and the distance values were consistent.

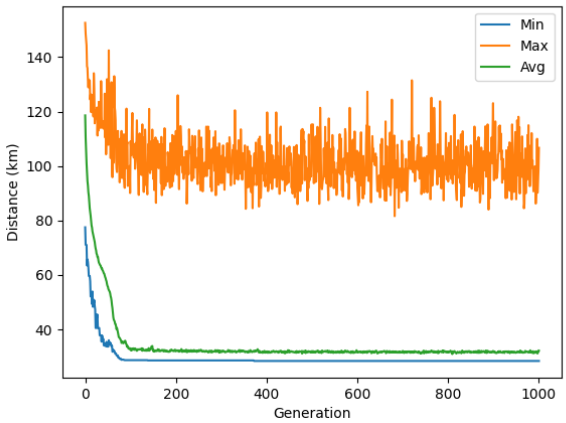
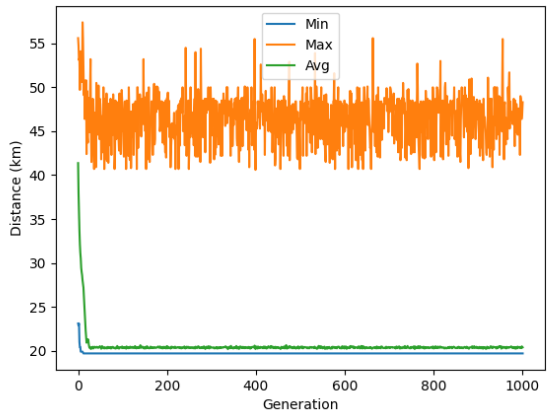


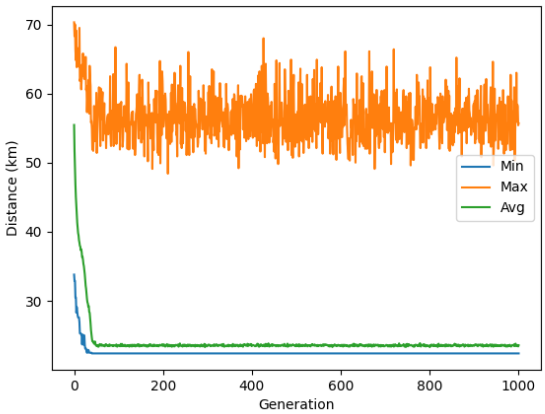
Figure 2: Maximum, Average and Minimum Distance with POP_SIZE = 1900 and NUM_GEN = 1000 for 30 EcoPoints.

After concluding that the best results were obtained with the parameters population size =

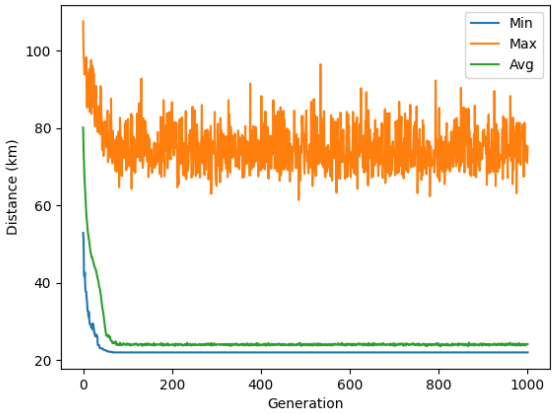
1900 and number of generations = 1000, we proceeded to changing the number of EcoPpints per route:



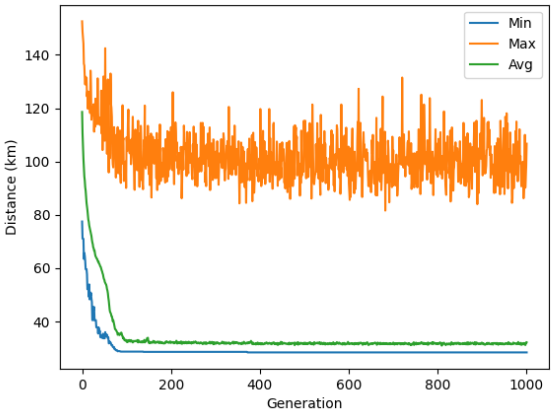
(a) 10 EcoPoints.



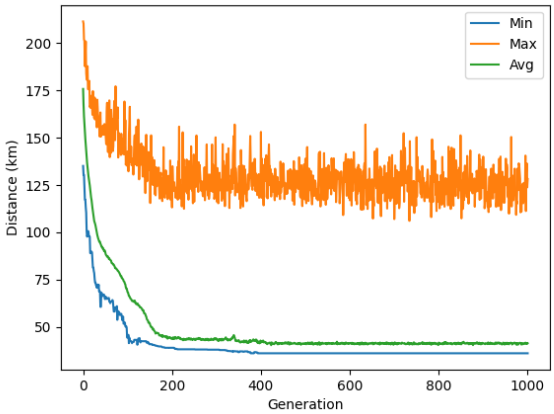
(b) 15 EcoPoints.



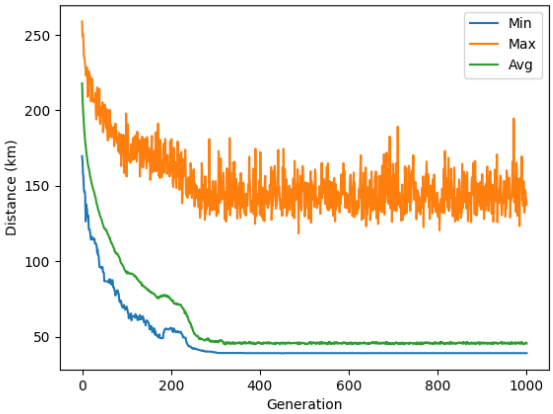
(c) 20 EcoPoints.



(d) 30 EcoPoints.



(e) 50 EcoPoints.



(f) 60 EcoPoints.

Figure 3: Maximum, Average and Minimum Distance with POP_SIZE = 1900 and NUM_GEN = 1000.

The best route determined for each amount of EcoPoints was the following:

10 Ecopoints:

Algorithm execution time:

34.49 seconds

Best route:

C, 4.1E77, 1.7E96, 0.7E74, 0.5E16, 1.7E38, 0.9E73, 4.8E65, 0.0E19, 0.3E48, 2.4E41, 0.3C, Total=17.40

15 Ecopoints:

Algorithm execution time:

40.23 seconds

Best route:

C, 4.0E79, 1.1E81, 0.7E55, 2.1E51, 0.8E13, 0.4E91, 2.3E94, 2.2E58, 0.6E45, 2.2E43, 2.4E10, 0.1E11, 1.0E25, 0.0E98, 0.1E68, 2.2C, Total=22.20

20 Ecopoints:

Algorithm execution time:

46.03 seconds

Best route:

C, 6.5E38, 1.1E95, 1.1E42, 0.4E35, 1.1E36, 0.0E70, 0.5E78, 0.2E71, 0.9E02, 0.7E21, 0.6E44, 0.8E18, 0.5E79, 2.0E72, 2.3E07, 1.9E65, 0.0E10, 0.0E47, 0.0E15, 2.9E66, 0.1C, Total=23.60

30 Ecopoints:

Algorithm execution time:

58.70 seconds

Best route:

C, 3.0E76, 3.3E16, 0.5E13, 0.7E32, 0.5E56, 0.4E50, 0.1E31, 0.0E08, 1.3E42, 1.4E83, 0.5E95, 1.1E40, 0.7E73, 1.2E57, 0.2E58, 1.2E53, 1.0E80, 0.2E21, 1.2E52, 0.3E96, 1.1E28, 0.5E69, 2.4E89, 1.1E07, 1.9E11, 0.1E49, 0.0E61, 0.3E48, 2.1E14, 0.7E66, 0.1C, Total=29.10

50 Ecopoints:

Algorithm execution time:

85.95 seconds

Best route:

C, 2.1E72, 1.9E79, 0.0E27, 0.5E18, 0.4E64, 1.0E81, 0.7E28, 1.5E82, 1.7E23, 0.6E33, 0.6E80, 0.2E21, 0.4E55, 2.0E88, 0.0E32, 0.0E22, 0.1E39, 0.6E71, 0.2E56, 0.2E99, 0.7E24, 0.1E91, 0.4E16, 0.5E02, 1.1E92, 1.5E54, 0.3E73, 1.7E86, 0.3E83, 0.0E93, 0.5E95, 1.1E38, 1.4E09,

0.6E58, 0.5E87, 0.8E45, 0.5E30, 0.4E01, 2.2E43, 2.5E15, 0.1E65, 0.0E49, 0.1E19, 0.1E47, 0.0E10, 0.9E25, 0.0E98, 0.1E68, 1.2E67, 0.7E41, 0.3C, Total=35.30

60 Ecopoints:

Algorithm execution time:

99.99 seconds

Best route:

C, 6.5E40, 1.4E93, 0.3E86, 0.3E94, 0.1E95, 1.1E38, 0.5E35, 1.6E56, 0.0E78, 0.3E04, 0.4E88, 0.1E22, 0.4E99, 0.3E71, 0.3E50, 0.1E91, 0.1E24, 0.0E08, 0.0E31, 0.0E36, 0.3E16, 0.5E02, 0.4E23, 0.7E52, 2.9E76, 3.0E14, 0.8E20, 0.8E59, 1.5E98, 0.1E68, 0.0E03, 0.6E48, 0.3E19, 0.0E61, 0.0E65, 0.0E49, 0.0E15, 0.0E47, 0.0E29, 0.2E26, 1.7E07, 1.0E34, 0.0E43, 1.8E30, 0.0E60, 0.1E12, 0.5E87, 0.6E54, 0.6E58, 0.2E57, 0.4E45, 1.7E82, 0.6E18, 0.5E79, 0.1E77, 1.0E33, 0.5E55, 0.7E80, 0.5E81, 1.0E64, 4.1C, Total=43.50

The next step was to run the code but now with all the 99 EcoPoints:

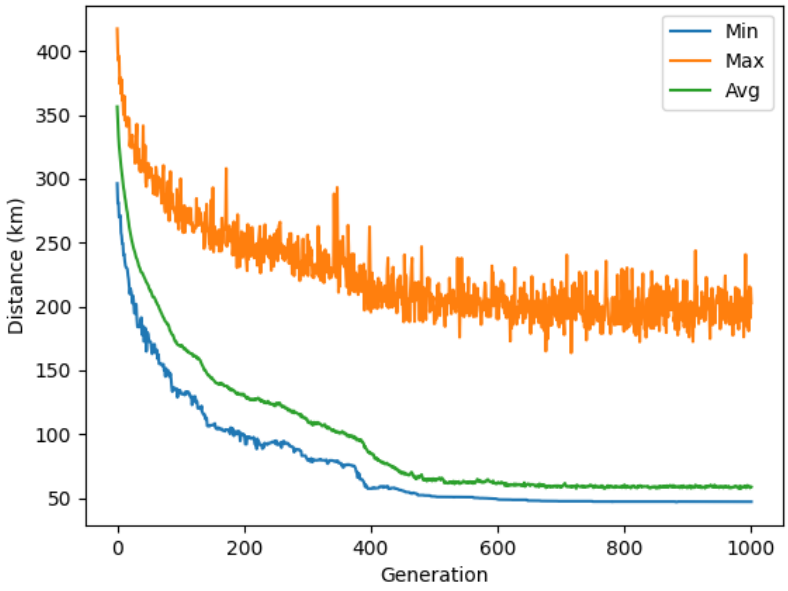


Figure 4: Maximum, Average and Minimum Distance with POP_SIZE = 1900 and NUM_GEN = 1000 for 99 EcoPoints.

By analysing the figure 4 above it is possible to see that the minimum value of the distance starts to stabilize from the 700th generation, which means that it was necessary at least 700 generation for the result to be consistent. The minimum value stabilizes around 40 to 50 km. The maximum of the distance tend to converge but not stabilise.

After running the code several times, we concluded that for the parameters selected this was the best solution for the 99 EcoPoints route:

Algorithm execution time:

155.72 seconds

Best route:

C, 3.7E07, 1.5E60, 0.0E30, 0.1E12, 0.5E87, 0.5E58, 0.2E57, 0.5E09, 0.9E53, 0.6E75, 0.7E73, 1.3E95, 0.1E94, 0.3E86, 0.3E93, 0.0E83, 1.2E05, 0.3E40, 0.3E38, 0.7E92, 0.6E62, 0.6E44, 0.5E28, 0.4E33, 0.1E69, 0.0E85, 0.3E81, 0.5E80, 0.2E21, 0.4E55, 0.2E23, 0.4E02, 0.7E96, 0.4E52, 0.9E16, 0.5E13, 0.0E71, 0.4E74, 0.3E04, 0.4E22, 0.0E32, 0.0E88, 0.2E51, 0.1E39, 0.3E99, 0.0E97, 0.1E56, 0.0E78, 0.4E50, 0.1E36, 0.0E91, 0.0E70, 0.0E08, 0.0E31, 0.0E24, 1.1E35, 0.4E42, 0.2E37, 1.4E54, 1.0E45, 0.5E01, 0.6E17, 1.1E89, 0.0E34, 0.0E43, 1.3E64, 0.5E82, 0.6E18, 0.5E27, 0.0E79, 0.1E77, 1.4E76, 1.4E72, 1.3E90, 0.8E84, 0.8E67, 0.3E06, 0.5E41, 0.3E66, 0.2E63, 0.4E59, 0.0E14, 1.5E98, 0.0E25, 0.1E68, 0.0E03, 0.6E48, 0.3E15, 0.0E61, 0.0E29, 0.0E10, 0.0E47, 0.0E49, 0.0E65, 0.0E19, 0.0E11, 0.8E46, 0.8E26, 3.1E20, 0.2C, Total=46.88

We initially tested the code but with the parameters POP_SIZE = 300 and NUM_GEN = 200 and every time we ran the code a different solution appeared with discrepant values (the best distance was oscillating between ≈ 130 km to ≈ 40 km) and that is why we concluded that was necessary to increase the population size and the number of generations. Although with the increase of those parameters the value of distance for the best route was not oscillating to much, but it was not the same every time we ran the code (there were small changes in every test made). After all, we had a trade off, the value of distance tend to stabilize with the increasing of population size and the number of generations but that would imply a greater execution time.

To see the evolution of the algorithm execution time with the number of EcoPoints per route, several tests were made and the results are presented in the figure (5) bellow:

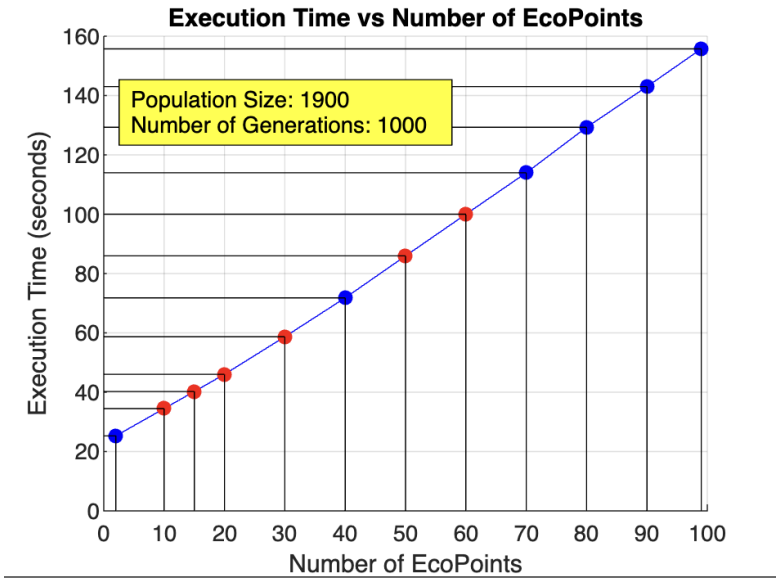


Figure 5: Execution Time vs Number of EcoPoints.

On the figure above (figure 5) the red dots represent the marks of 10, 15, 20, 30, 50 and 60 EcoPoints and the blue ones represent other values of number os EcoPoints just to give a general idea of the minimum and maximum algorithm execution time.

As it is observable, the ratio between the number of EcoPoints per route and the algorithm execution time is almost linear. This demonstrates the scalability of the GA.

With the population size and number of generations fixed in 1900 and 1000 respectively, it is possible to see by the plot that for 99 EcoPoints the algorithm takes around 160 seconds (or $\approx$ 2 mins 40 seconds) to determine the best route. If we increased the values of those parameters it would take more time and would be overkill to gain just a few meters in the final solution.

4 Conclusion

Overall we have found the developed Genetic Algorithm to be an adequate solution for this TSP-like problem.

Through the utilization of evolutionary principles such as selection, crossover, and mutation, the GA emulates natural selection, efficiently exploring the search space, and converging towards the optimal ecopoint route . This is done whilst offering scalability for the number of ecopints in each route, allowing for a feasible (if not optimal) solution for a real world problem.

5 Annex: Function Optimization (Lab 7)

For Lab 7, we developed solutions using Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) in order to determine which points $X1$ and $X2$ maximize the function $f1$ described below:

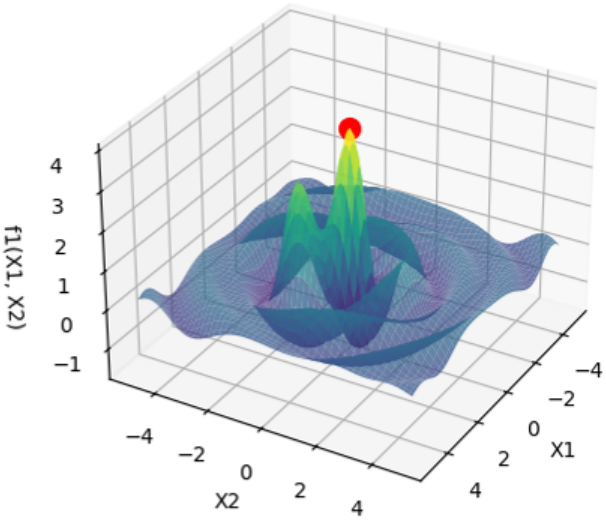
$$\begin{aligned} Z1 &= \sqrt{X1^2 + X2^2} \\ Z2 &= \sqrt{(X1-1)^2 + (X2+1)^2} \\ f1 &= (sin(4 \cdot Z1)/Z1) + (sin(2.5 \cdot Z2)/Z2) \end{aligned} \tag{1}$$

Where as in the project each individual was an ordered list, for this problem an individual will be a set of two real values (X1 and X2) so the approach to building the algorithms was slightly different.

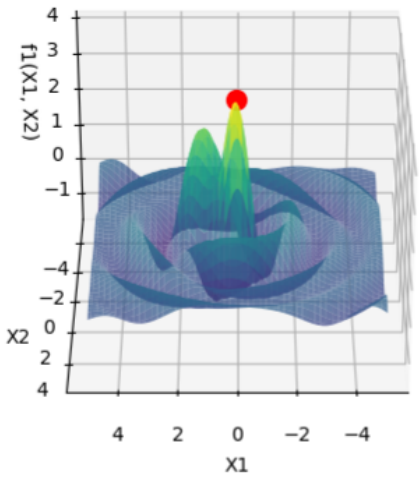
We've seen how GAs explore the search space through the evolution of populations using genetic operators for mating, mutation and selection.

PSO, whilst also a nature-inspired algorithm, focuses on the collective behavior of individuals (here called particles) to converge towards an optimal solution. These particles adjust their positions based on their own best-known position and the global best-known position found by the entire swarm. The movement of particles is guided by velocity vectors, which are updated iteratively according to cognitive and social parameters. The weight of these parameters can be tuned so as to determine whether a particle is more independent in its search or more pressured to conform to the behaviour exhibited by its neighbors.

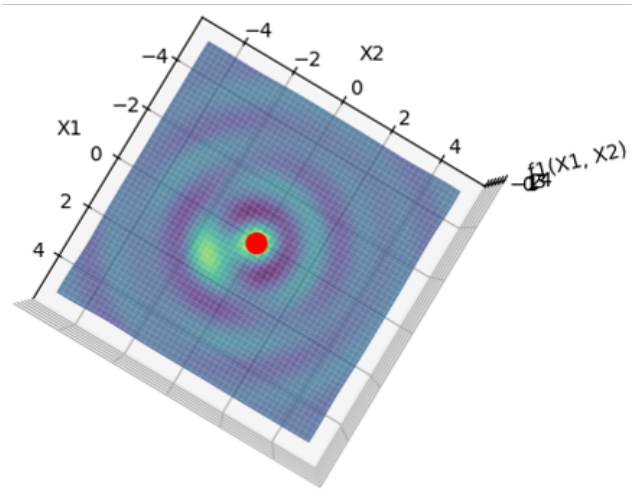
Having run both implementations, the maximum of $f1(X1, X2)$ was found to be approximately 3.7876807698 where $X1 \approx 0.05826$ and $X2 \approx -0.05826$. A 3D Plot represents the shape on the function and pinpoint the coordinates of the solution:



(a) $f1(X1, X2)$ (Isometric View).



(b) $f1(X1, X2)$ (Side View).



(c) $f1(X1, X2)$ (Top View).

Figure 6: $f1(X1, X2)$ with Maxima annotated in Red.

The plots below track how each algorithm evolved:

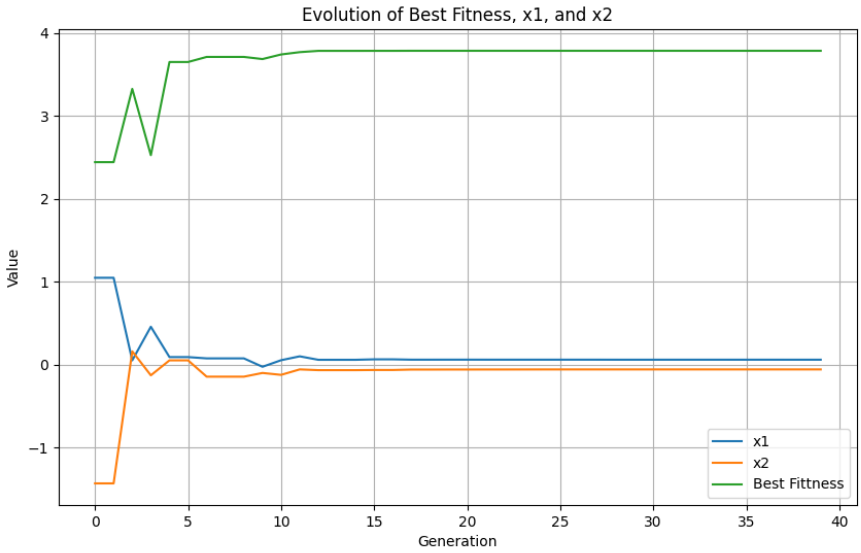


Figure 7: GA Evolution.

In the case of GA, we determined that for a population size of 50, 40 generations was sufficient to achieve convergence. The population size has a great effect on the performance of the GA. For low population sizes (say 20) the number of generations must be substantially increased or a more moderate increase along with higher the crossover and mutation ratio to ensure selectivity and variation, which help the algorithm converge. The CPU time for the GA was 0.01 seconds.

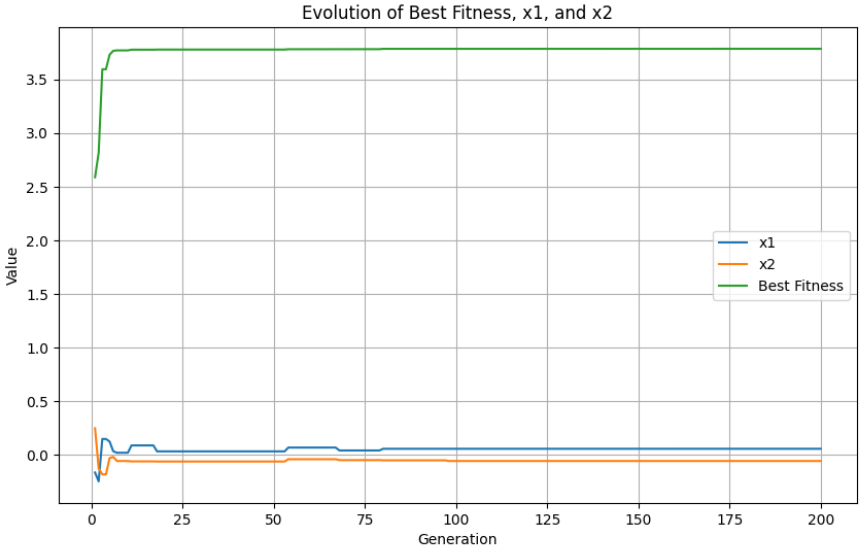


Figure 8: PSO Evolution.

For PSO, best results were obtained by reducing the max speed of the particles (from initial 3 to 2) and putting more weight on neighborhood results than personal results (2.5 vs 1.5).

Compared to GA, PSO required a larger population size (50 vs 100) and more generations (40 vs 200) to achieve similar results and even then the results produced by PSO were far less consistent. Additionally, PSO was more computationally expensive with a CPU time of 0.12 seconds.

6 References

Git Hub Repository:

https://github.com/manuel-passadouro/CIIC

ChatGPT search engine:

https://chatgpt.com/

Lecture Slides:

https://fenix.tecnico.ulisboa.pt/downloadFile/282093452120783/7-StudenCI IoT EvolutionaryComputation.pdf

DEAP Documentation:

https://deap.readthedocs.io/en/master/index.html

Genetic Algorithm: The Travelling Salesman Problem via Python, DEAP:

https://medium.com/@pasdan/genetic-algorithm-the-travelling-salesman-problem-via-python-deap-f238e0dd1a73

The Genetic Algorithm and The Travelling Salesman Problem(TSP):

https://itnext.io/the-genetic-algorithm-and-the-travelling-salesman-problem-tsp-31dfa57f3b62