NOVA IMS Information Management School

COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

Using Genetic Algorithms to Enhance Song Recognition

Group: Teletubbies

Beatriz Xavier (20230987); **Mariana Cabral** (20230532);
Sofia Pereira (20230568); **Susana Pires** (20230540)

Prof. Berfin Sakallioglu | Prof. Leonardo Vanneschi

https://github.com/sofiacper/CIFO_Project

June 2024

CONTENTS

1. INTRODUCTION

For music enthusiasts and researchers alike, accurately recognising and matching melodies to iconic compositions poses a significant challenge.

This project introduces *Genetic Algorithms* (GAs) as a novel approach to the task of song recognition by experimenting with a diversity of approaches and evaluating their performance on matching a given target song.

1.1 DIVISION OF LABOR

Given that the scope of the chosen project closely resembles the *Traveling Salesperson Problem*, the team's focus was not on developing an implementation from scratch, but in adapting the *classes* developed during the labs and then implementing and comparing different operators (crossovers, mutations, selections, and initializations) that would optimize the solution of the problem.

Table I details specific tasks assigned to each group member and the general division of labour.

2. PROBLEM DEFINITION

Given the problem's objective of evolving individuals to closely resemble a specific target and considering that GA's require individuals to be represented as strings of constant length, all potential solutions were constructed around the characteristics of the target itself. Since any song could be encoded as sequences of MIDI notes[1], the group searched for a representation already codified, finding the *Super Mario Brothers Game*(1) theme song, a sequence of 312 notes.

Considering this, the **individuals** were defined as a melody, i.e., a sequence of notes stored in a string of the same size as the target (312) that could contain any value in the MIDI range. The **search space** was composed of all melodies that fall into these same characteristics. Finally, the problem consisted of **minimizing** the distance between a specific melody and the target, which was measured using a combined **fitness function** that considered the weighted sum of the *Euclidean* and *Hamming* distances(2).

3. DEVELOPMENT OF GENETIC ALGORITHM

The proposed algorithm aimed to evaluate and compare different evolution behaviours, experimenting with a diversity of initialization and selection methods and also cross-over, mutation operators.

3.1 SELECTION OPERATORS

The selection algorithms used for this project were the *Ranking*, *Tournament*, and *Fitness Proportionate Selection* (FPS), also known as *Roulette Wheel*. Of these algorithms, only the first had not been implemented in lab classes, but all were discussed in theoretical classes(3). Since it was determined that the main focus of the project was to implement and test the effects of different variators, these three selection algorithms were used as a baseline to further experiments.

3.2 CROSS-OVER OPERATORS

Crossovers are essential for genetic algorithms, as they combine the genetic information of parent solutions to produce offspring with potentially better fitness. This process promotes genetic diversity and helps the algorithm explore a wider solution space. In this way, crossovers help avoid local optima by introducing new variations into the population. Even though *Cycle Crossover* and *Partially Mapped crossover* were implemented in class, the group decided not to include them in the project since the problem's representation had repetition. The crossovers tested for this project were:

***Single Point Crossover/Two-point Crossover*:** single point crossover randomly generates a crossover point which determines the points for information sharing between parents to form children. Two-point crossover is a specific case of n-point crossover technique, in which there is not only one but two crossover points.

***Uniform Crossover*:** each gene is independently swapped between the two parent organisms based on a random probability (in our case, smaller than 0.5, so each gene would be chosen with equal probability), resulting in two new offspring that inherit a mix of genes from both parents.

***Order One Crossover*:** a random subsequence is chosen from each parent and directly copied into the corresponding positions of the offspring. The rest of the offspring's genes are filled in the order they appear in the other parent, ensuring no duplicate genes and maintaining the genetic sequence.

***Arithmetic Crossover*:** a single random value, alpha, is generated from a uniform distribution between 0 and 1 and is used to blend the genes of the two parents. Each gene in the offspring is a weighted average of the corresponding genes from both parents, with each offspring favouring the genes of each parent.

---

[1] **MIDI notes:** numerical values representing musical notes across different platforms and systems, being a widely recognized and accepted standard in the music industry and digital music technology Each *MIDI note* includes pitch, velocity and duration values. For simplicity, it was considered the pitch range from 0 to 127.

***Blend Crossover/Blend Alpha Crossover*:** Blend crossover randomly selects values within the range defined by the parent genes, while Blend Alpha extends this range using an exploration intensity parameter, allowing for wider exploration.

3.3 MUTATION OPERATORS

When it comes to the mutation operators selected to be studied, it was considered that, since mutation has a low probability of occurring, these operators would possibly not have a big impact on model performance. Nevertheless, the following operators were tested:

Swap Mutation: this mutation randomly selects two positions of the individual and swaps their values.

Random Resetting: this operator selects *n* indices and replaces their values with a random value from a valid set of numbers. In the context of this problem, 5 indices were selected to be changed in the MIDI range.

***Inversion Mutation / Reverse Sequence Mutation* (RSM):** this type of mutation selects a subset of genes and inverts their order.

***Scramble Mutation / Partial Shuffle Mutation* (PSM):** much like the operator above, this mutation selects a subset of genes and randomly shuffles them.

***Centre Inverse Mutation* (CIM):** this operator divides the gene into two parts and reverses each part separately.

3.4 GEOMETRIC OPERATORS

Considering that *Geometric Operators* were introduced for continuous optimization, this approach aimed to assess their effectiveness in addressing the discrete optimization challenge. As the individuals were initially generated as integer strings, the only necessary adjustment was to guarantee that the solutions remained within the valid set, which was only verified in mutation, preventing disruptive cases. Therefore, this was not problematic during cross-over, as the linear combination always yielded integers within the parents or in the worst-case scenario, one of the parent's genes was employed.

Additionally, considering the specific parameters of these operators, it is worth mentioning the *mutation_step*, which was fixed at the value of 1 for simplification. This decision was made to emphasize generating substantial changes in the mutated gene, given that mutations are infrequent. However, if this method proves effective for the problem, further exploration should be conducted.

3.5 INITIALIZATIONS

For the final steps of our project, it was decided to implement different initializations, to study their impact on our model and hopefully improve its performance.

The first initialization consisted of the one studied in class, which is a *random choice* within the MIDI note range [0, 127].

The second initialization, *melody_rules*, begins with the central C (MIDI range value of 60), a commonly used starting note for countless musical compositions and within the range of several musical instruments. After this, it performs a form of random walk, with the size of each step selected from the interval [-*max_step*, *max_step*], which is a function parameter. This avoids excessively large jumps between notes. The function also constrains each note to lie within the MIDI note range.

The final initialization, *sobol_sequence,* generates an array of values using a *Sobol' Sequence*, which is known for generating quasi-random sequences and having a more uniform sampling compared to purely random sequences. This method aims to encourage diversity in the initial population, having it more evenly distributed. The function initializes a *Sobol' Sequence* generator for one dimension. *Sobol' Sequences* are based on binary principles, the size of the problem's sequence based on powers of 2 need to be determined. This means calculating the base-2 logarithm of the desired length. Doing this means essentially finding out how many bits it needs to represent that length in binary. Then, rounding that number to the nearest whole number gives us a power of 2. These values are then flattened and scaled to the range [0, 127].

4. PERFORMANCE EVALUATION

Considering all kinds of possibilities that exist in terms of combinations of operators, the evaluation of the performance of our GA was divided into different phases.

4.1 1ST PHASE – TESTING OUT ALL OPERATORS AGAINST A BASELINE MODEL

Looking to explore the impact each operator has on the chosen problem, this phase focuses on comparing different methods against a base model (which consisted of tournament selection, swap mutation and single-point crossover with random initialization and elitism). From there, two other studies were done: a **mutation study** (Figure 1, Figure 3), focused on changing the mutation operator from baseline model and testing them for all selection algorithms, and a **crossover study** (Figure 2, Figure 4) following this same structure. For comparison and consistency purposes, a population of size 500 was selected and evolved for 300 generations. Each algorithm was run through 10 iterations and

a median of its fitness through each generation was used to assess the performance. Looking also to assess the impact of defining different fitness functions, these experiments were implemented for two fitness functions: using only *Euclidean* distance and calculating its weighted average with the *Hamming* distance (in a 50/50 weight).

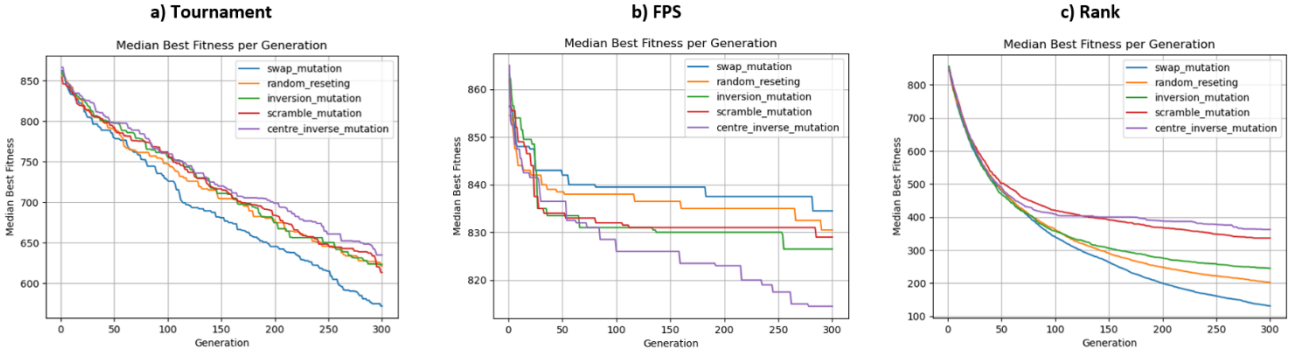

Figure 1 – Mutation study with selection algorithms for Euclidean fitness.

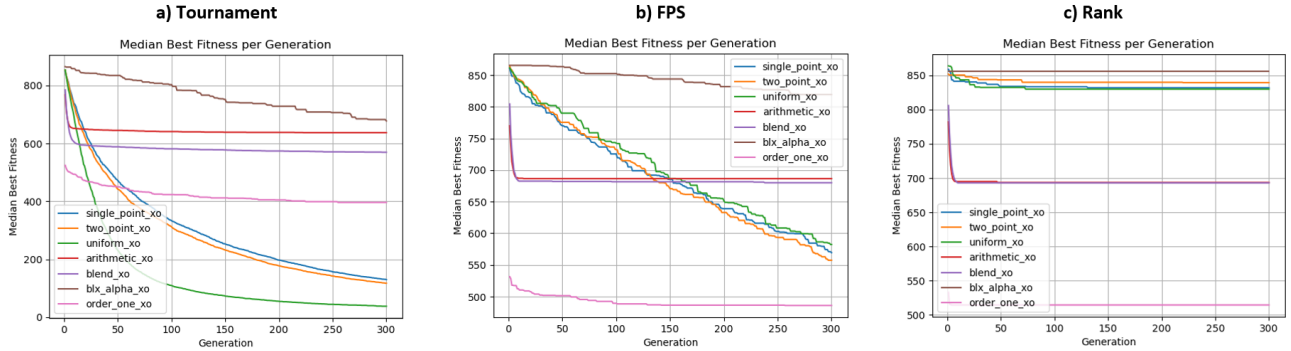

Figure 2 – Crossover study with selection algorithms for Euclidean fitness.

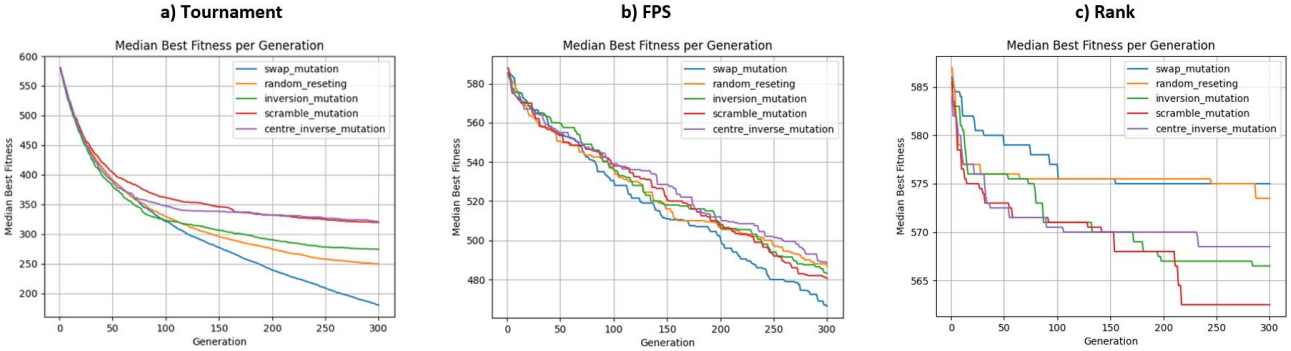

Figure 3 – Mutation study with selection algorithms for weighted fitness.

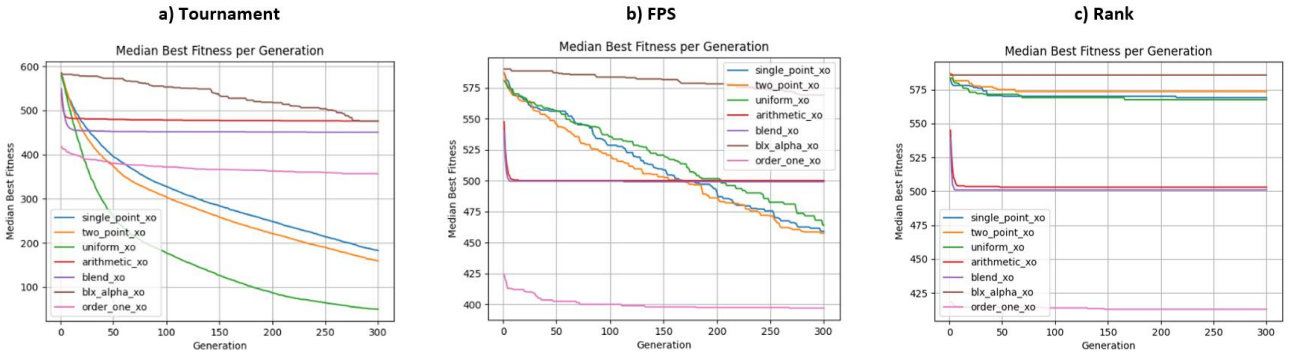

Figure 4 – Crossover study with selection algorithms for weighted fitness.

The same experiment was done separately for the *Geometric Operators* (Figure 5), to test the operators' performance with different selection algorithms and fitness.

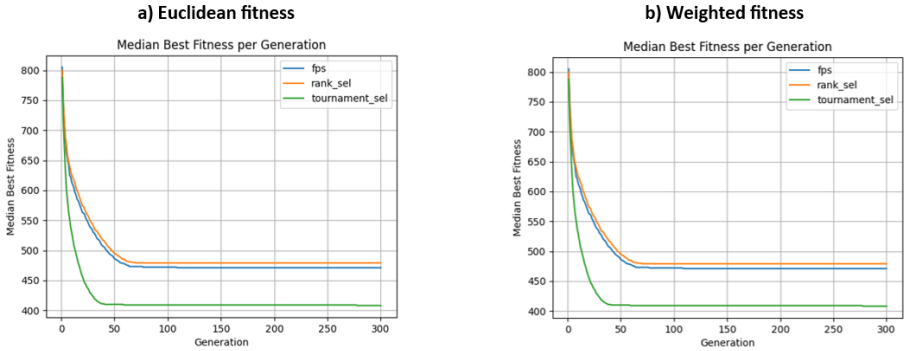

Figure 5 – Selection study for *Geometric Operators*.

Looking at the scores obtained for the Median Best Fitness per generation for each experiment, it is notorious that for both fitness and the *Geometric Operators*, the best selection algorithm is ***Tournament Selection***, as it reaches substantially lower values of fitness when compared to other methods. Focusing on this selection's graphs (Figure 1-a, Figure 3-a), *Random Resetting* and *Uniform Mutations* achieve the best performance for both fitness. When it comes to the crossover algorithms (Figure 2-a, Figure 4-a), *Uniform* and *Two-point Crossovers* are the algorithms that lead to the lowest median fitness value for both fitness.

The general best combinations are ***Tournament Selection*** with ***Random Resetting*** and ***Uniform Mutation***, and this same selection with uniform and ***Two-point Crossover***. ***Euclidean*** distance (Figure 2-a) achieved a slightly lower score than using the weighted average fitness (Figure 4-a), but both implementations are kept for further analysis.

It is important to note the impact that different operators have on the performance and convergence of the model, being the operators selected the ones that show a convergence toward lower values of fitness and so, the ones chosen for further studies.

4.2 2ND PHASE – CHOOSING THE BEST COMBINATIONS

Different combinations of the previously selected operators were tested against each other, looking to assess what pair would achieve the best general performance. To achieve better consistency for all experiments, it was kept the same values for the number of runs (=10), population size (=500), number of generations to evolve (=300) and elitism (=True).

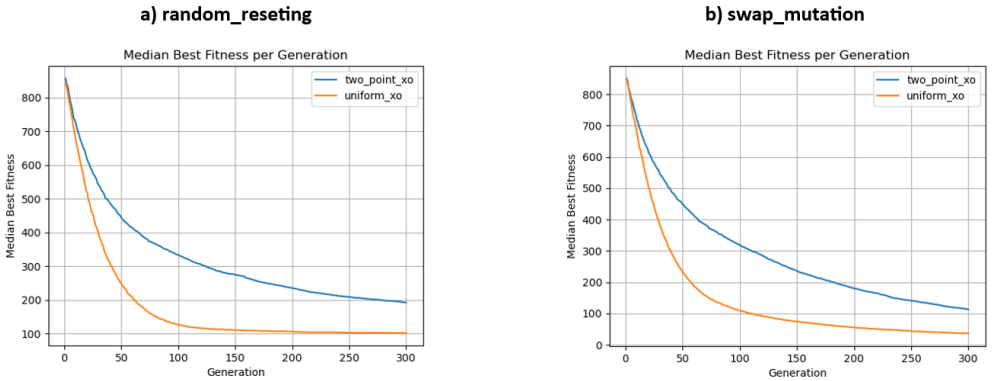

Figure 6 – Mutation and crossover study for Euclidean fitness.

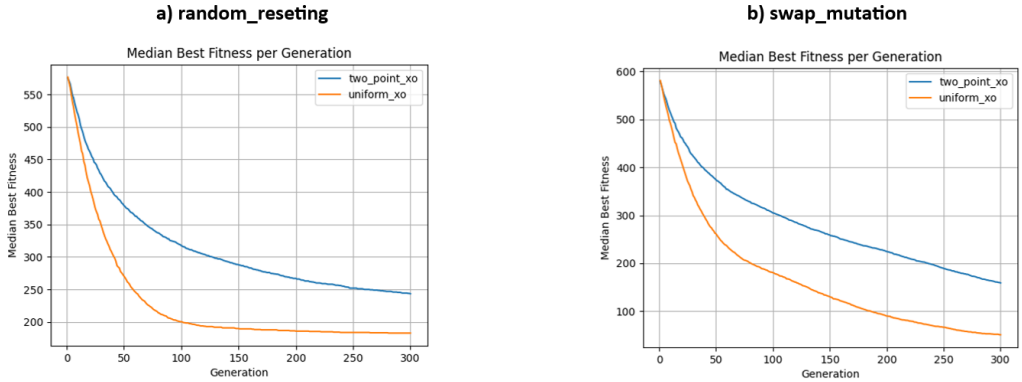

Figure 7 – Mutation and crossover study for weighted fitness.

From the fitness scores acquired in these experiments (Figure 6, Figure 7), it's possible to determine that the combination of ***Swap Mutation*** with Uniform Crossover is giving the best results for both fitness approaches (reaching below the fitness value of 100), considering the previously fixed parameters.

With these results, it's possible to settle more parameters, in order to go some more steps further into this comprehensive analysis. For the next phase the study of both fitness will be continued.

4.3 3RD PHASE – OPTIMIZING PARAMETERS

Finally, it was decided to conduct a comprehensive analysis of the genetic algorithm's performance over 500 generations, using *Tournament Selection*. The experiments were run by testing for different parameters: population sizes of 250, 500 and 1000 individuals, crossover rates of 1, 0.9 and 0.5 (for *Uniform Crossover*), and mutation rates of 0, 0.2 and 0.5 (for *Swap Mutation*). The goal was to identify the best parameter settings. Furthermore, it was also evaluated the impact of different initializations, both with and without elitism.

Regarding population size (Figure 8), the best fitness score was achieved at a size of **1000 individuals**, although the difference in fitness between 1000 and 500 individuals is smaller than between 500 and 250.

a) Euclidean fitness **b) Weighted fitness**

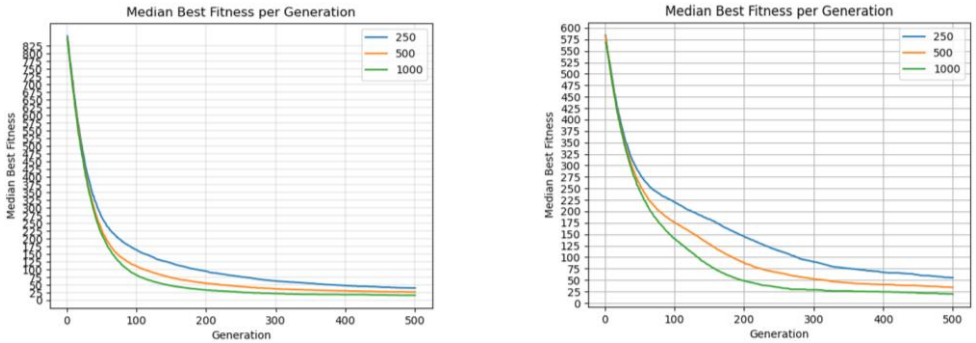

Figure 8 – Population size study for both fitness.

For crossover rates, the best fitness score of 14 occurred at a **rate of 0.9/1** (Figure 9). These results were expected, since a high crossover rate promotes diversity, potentially leading to the discovery of better solutions. When considering mutation rates, the best fitness score of 10 was found for a **mutation rate of 0.5** (Figure 10), which shows that a more disruptive approach can, sometimes, be advantageous. Since mutation is considered an "innovator operator", it can help explore new regions of the search space, therefore also increasing diversity.

a) Euclidean fitness (≈14) **b) Weighted fitness** (≈18)

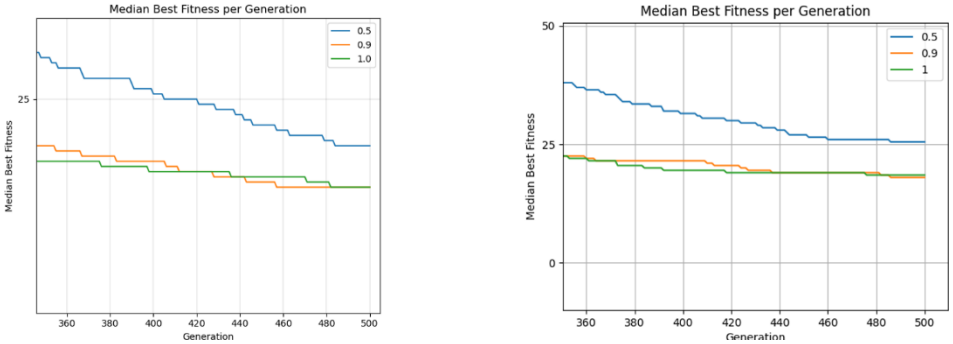

Figure 9 – Crossover rate study for both fitness.

a) Euclidean fitness (≈11) **b) Weighted fitness** (≈10)

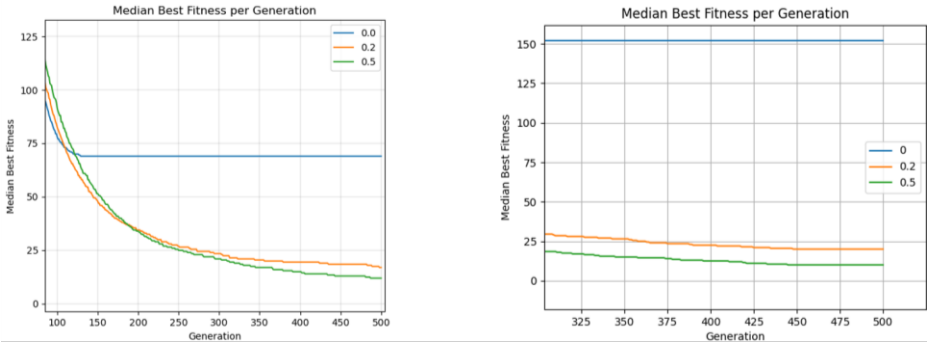

Figure 10 – Mutation rate study for both fitness.

Among the initializations tested, *random search* proved to be the most effective approach, achieving a fitness score of 10 with elitism (Figure 11). The *melody_rules* initialization (Figure 12) may have limited initial diversity since it starts on a specific note and performs a random walk with small steps. On the other hand, the *sobol_sequence* initialization (Figure 13) might not have resulted in better fitness due to the specific nature of the target melody, which is not evenly distributed across the musical scale. Surprisingly, it was observed that **omitting elitism** led to similar results. This can happen because the algorithm becomes overly conservative, especially when only retaining one individual, inhibiting exploration and leading to loss of diversity.

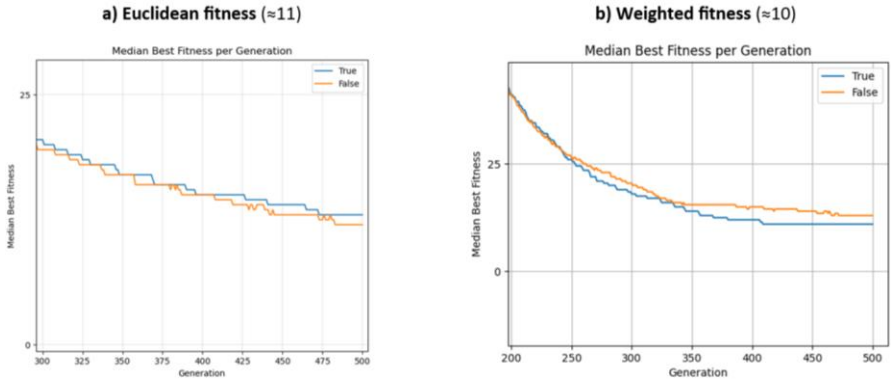

Figure 11 – Random Initialization study with elitism (=True/False) for both fitness.

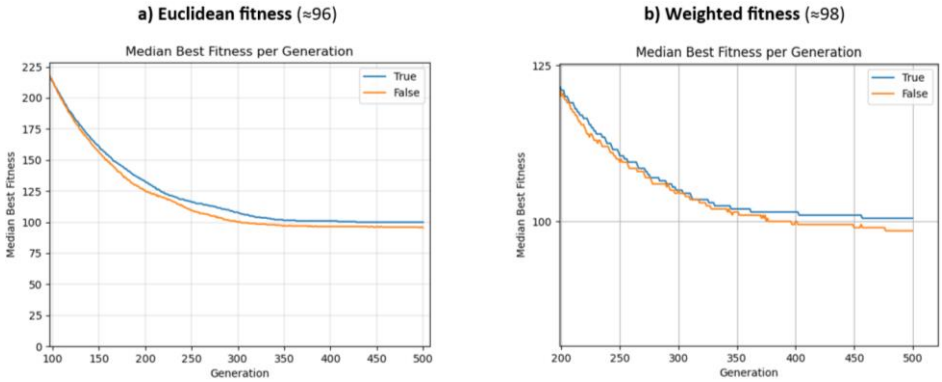

Figure 12 – Melody Rules Initialization study with elitism (=True/False) for both fitness.

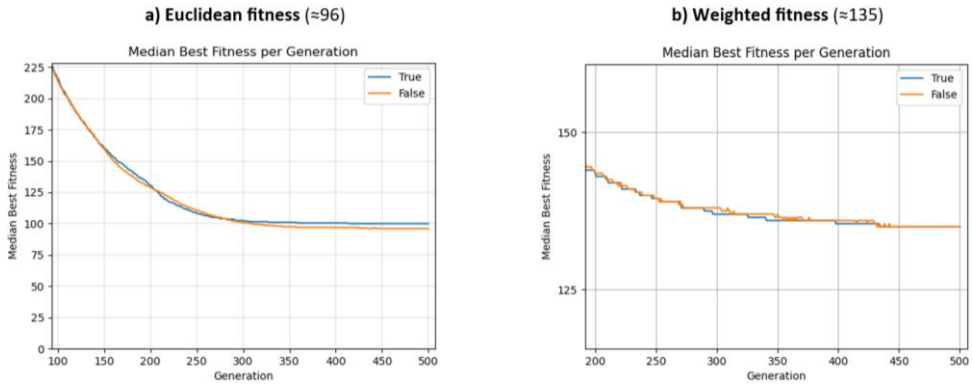

Figure 13 – Sobol Sequence Initialization study with elitism (=True/False) for both fitness.

5. CONCLUSION

The exploration yielded promising results, despite the optimization algorithm failing to reach the global optimum, in the final stages of the tunning it managed to reach a solution with a **fitness score of 10**, for the weighted fitness, noticing that the worst-case scenario was a fitness in the order of 0.5*312+0.5 L, where L places a large number.

While these findings were significant, they didn´t mark the conclusion of this project. Indeed, numerous additional explorations could be pursued to further refine our understanding. However, it was pertinent to acknowledge

the constraints posed by computational power. Given these limitations, the group considered the achieved solution satisfactory.

Looking ahead, the group also identified several suggestions for improvement and future work. First, regarding the fitness function, the following question was raised: how do the weights in the fitness influence the convergence/evolutionary behaviour? So, it was distinguished that exploring and evaluating other weight ponderations could have provided additional insights.

Secondly, in terms of the optimization and tuning strategy, considering that it followed a specific sequence of steps, investigating alternative orders/approaches and assessing their impact on the final outcomes could have been advantageous. For instance, what if it was tested the impact of the initializations in the beginning? Or what if it were decided to tailor an optimization according to a certain initialization? Moreover, testing a wider range of parameter value combinations might have offered deeper insights into the optimization process.

Lastly, concerning the evaluation strategy, although a plot illustrating the median fitness over 10 runs was generated, it was acknowledged that conducting this analysis for a statistically significant number, such as 30, would have been ideal. Additionally, the inclusion of hypothesis tests to evaluate the significance of the observed differences in results could have strengthened the evaluation process.

6. ANNEXES

Table I – Distribution of tasks

Group member	Task
Beatriz	Crossovers and Initializations
Mariana	Geometric operators and Fitness function
Sofia	Mutations
Susana	Selection algorithms
Everyone	Ideas, Experiments, Conclusions, Report

7. REFERENCES

1. CodyJung, 262588213843476. GitHub Gist. [cited 2024 May 31]. Super Mario Brothers sheet music using MIDI values. Available from: https://gist.github.com/CodyJung/2315618

2. Gohrani K. Different Types of Distance Metrics used in Machine Learning [Internet]. Medium. 2019 [cited 2024 May 31]. Available from: https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7

3. Vanneschi L, Silva S. Lectures on Intelligent Systems [Internet]. Cham: Springer International Publishing; 2023 [cited 2024 May 31]. (Natural Computing Series). Available from: https://link.springer.com/10.1007/978-3-031-17922-8