COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

Master in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

Music Festival Lineup Optimization

Group O

Ana Margarida Valente, 20240936

Luana Rocha, 20240111

Pedro Costa, 2022121

Susana Reis, 20240567

Github link: CIFO-Project

Spring Semester 2024-2025

TABLE OF CONTENTS

1. Introduction 1
2. Optimization problem 1
3. Detailed Description of Implemented Selection and Genetic Operator 2
   3.1 Selection Mechanisms 2
   3.2 Crossover Operators 3
   3.3 Mutation Operators 3
4. Performance Analysis 3
   4.1 Operator Selection and Influence on Convergence 3
   4.2 Fine-Tuning Operator Probabilities 5
5. Conclusion 5
6. Appendix 6

1. INTRODUCTION

In this project we are given the task of designing a multi-stage music festival schedule using a genetic algorithm. Rather than manually navigating the overwhelming space of possible artist arrangements, we rely on evolutionary principles to guide the search for high-quality solutions. We will take into consideration musical genre diversity, overlapping audiences and artists popularity.

A general hypothesis of optimization is that the set of all possible solutions is so large that it is impossible to enumerate all of them, looking for the best one(s). And this is why 'intelligent' algorithms are generally in demand for solving optimization problems (Vanneschi & Silva, 2023). Guided by this principle, we explored different algorithm configurations to understand their impact on performance. By adjusting components like selection, crossover, mutation, and elitism, the project examines how the genetic algorithm evolves solutions over time and which setups tend to produce better schedules.

2. OPTIMIZATION PROBLEM

The **goal** of this project is to optimize the scheduling of artists across 5 stages and 7 time slots during a music festival in the most balanced and interesting way possible. Each artist is associated with a popularity score, a genre, and a conflict score that measures fanbase overlap with other artists. The optimization problem seeks to generate festival lineups that simultaneously maximize the popularity of artists performing in prime (final) slots, increase genre diversity within each time slot, and minimize the scheduling of conflicting artists at the same time, Figura 1 illustrates the balance solution we seek. These three objectives are treated as equally important and normalized to ensure balanced evaluation. A valid solution must assign every artist to exactly one unique stage and time slot, ensuring no omissions or repetitions occur in the final lineup.

Each **individual** represents a complete festival lineup. This is structured as a list of 35 artist IDs, corresponding to the total number of slots (5 stages × 7 time slots per stage). Each position of the list directly maps to a specific stage and time slot using a fixed ordering. So, the first 7 positions correspond to the 7 slots of stage 1 and the next 7 are the 7 slots of stage 2 and so on.

This lists format is both efficient and compatible with genetic algorithm operations like crossover and mutation. It simplifies constraint checks such as avoiding duplicate bookings, and it allows for straightforward computation of evaluation metrics like genre diversity and prime-time popularity. It closely mirrors the real-world structure of a festival schedule while remaining suitable to the mechanics of a genetic algorithm.

The **search space** for this problem is extremely large due to the vast number of possible ways to assign artists to slots across stages. It consists of 35! unique permutations of artist assignments to stage/slot positions, approximately 1.03×10^{40} possible line-ups. Exploring all possible combinations exhaustively wouldn't be practical. To address this, a genetic algorithm is used to navigate the solution space and discover high quality lineups without evaluating every possible configuration.

The **fitness function** evaluates how good each solution is based on a few key points: making sure artists with overlapping fan bases don't play at the same time (so the audience doesn't have to choose), making sure the most popular artists are scheduled during prime time and encouraging genre diversity

across the stages and time blocks. The end goal is to find a lineup that strikes a good balance between all these elements and creates a better overall festival experience.

First, it calculates the highest possible popularity by looking at the most popular artists and assuming they all play in the prime slots (the last time slot on each stage). This gives a benchmark to compare against. Then, it measures how many different genres appear in each time slot and compares this to the maximum possible diversity, which is effectively restricted to 5 due to the constraint of having only 5 stages, even though there are 6 genres available (Rock, Electronic, Jazz, Classical, Pop and Hip-Hop). This means that, at most, 5 unique genres can appear simultaneously in a single time slot. Next, it estimates the worst-case conflict scenario, where all conflicting artists are scheduled at the same time, to create an upper limit for conflicts.

For the given lineup, the function calculates the actual popularity in prime slots, the genre diversity across the festival, and the total conflict based on a conflict matrix that shows which artists share fanbases. Each of these values is then normalized by the benchmarks calculated earlier, so they're on a comparable scale.

Finally, the fitness score combines these three normalized values by averaging the popularity, the diversity, and the inverse of the conflict (since less conflict is better). All factors are considered equally to ensure a balanced evaluation. This balanced score helps guide the search toward lineups that not only feature popular artists at prime slots but also ensure a good mix of genres and minimize audience conflicts.

$$fitness\_score = (norm\_prime\_pop + norm\_genre\_div + (1 - norm\_conflict)) / 3$$

3. DETAILED DESCRIPTION OF IMPLEMENTED SELECTION AND GENETIC OPERATOR

We used different types of selection, mutation and crossover operators to guide the genetic algorithm towards better festival lineups over generations.

3.1 Selection Mechanisms

We implemented two selection methods, where the parameters were kept fixed, to choose which individuals get to reproduce.

Tournament Selection: We randomly pick a small group (k = 3 individuals) from the population and select the best one based on fitness. This method adds a bit of randomness but still favors stronger solutions, helping avoid getting stuck too early in local optimum.

Ranking Selection: Here, individuals are sorted by fitness and given probabilities based on their rank rather than their raw score. This smooths out extreme fitness differences and gives lower-ranked individuals a small but fair chance of being selected, keeping the population diverse. The selection pressure (pressure = 1.7) controls how strongly higher-ranked individuals are favored.

3.2 Crossover Operators

Order Crossover (OX): A segment is copied from one parent, and the rest of the child is filled with genes from the other parent in their original order, skipping duplicates. This maintains the relative ordering of elements and ensures valid lineups. (Figura 2)

Partially Mapped Crossover (PMX): A segment is exchanged between parents, and a mapping process is used to resolve conflicts. This guarantees that each artist appears exactly once and preserves some positional information from both parents. (Figura 3)

3.3 Mutation Operators

Insertion Mutation: One artist is removed from a random position and reinserted at a different random position. In the example shown, gene 18 was moved from index 17 to index 28. (Figura 4)

Prime Slot Mutation: An artist in a prime slot (last slot of a stage) is swapped with one in a non-prime slot. This directly affects the objective of maximizing popularity in prime positions. In the example, the indices 27 (artist 28, prime slot on stage 4) and 14 (artist 15, slot 1, non-prime, on stage 3) were swapped. (Figura 5)

Slot Shuffle Mutation: A single time slot is selected, and the artists scheduled in that slot across all stages are shuffled. This increases variation without disrupting the overall structure. Lastly shown, slot 7 was shuffled across stages. Indices affected: [6, 13, 20, 27, 34] (Figura 6)

4. PERFORMANCE ANALYSIS

To evaluate the effectiveness of different configurations of our Genetic Algorithm (GA), we structured our experimentation in two phases. The first phase aimed to understand the influence of different genetic operators: crossover, mutation and selection methods, and the inclusion of elitism, on the convergence and performance of the algorithm. The second phase focused on fine-tuning the probabilities of these operators once the best-performing types had been identified. Success was measured by the fitness score assigned to each lineup, which ranges from 0 to 1.

4.1 Operator Selection and Influence on Convergence

In the initial testing phase, we set both the population size and number of generations to 100. This provided a computationally efficient way to compare operator configurations without significant training time. We tested 24 different combinations, resulting from:

- Two selection methods (tournament and ranking),
- Two crossover types (order crossover and partially mapped crossover),
- Three mutation types (inversion mutation, prime slot mutation and slot shuffle mutation),
- Each with and without elitism (True or False).

Each configuration was executed 30 times to reduce the effect of random variation and enable statistically reliable conclusions. The performance of each operator type was evaluated using plots that tracked fitness progression across generations.

We tested the impact of having Elitism set as True or False, this means that if elitism is True allows the best individuals of a generation to be preserved for the next generation, without any alteration. This leads to more consistent progress and faster convergence toward optimal solutions. As we can see from (Figura 7) when Elitism=True the mean or median of the fitness function performs better.

Next, we isolated the impact of Crossover and Mutation (Figura 8 and Figura 9). For crossover PMX consistently outperformed OX, producing higher average fitness scores, likely due to its ability to preserve relative ordering in permutations. On the other hand, in the Mutation graph the performance of mutation strategies varies more across generations. Initially, all three — insertion, prime, and shuffle — show similar growth in fitness. However, in Figure 1, after generation 40, insertion and prime continue to improve, while shuffle stabilizes. From generation 70 onwards, the insertion mutation demonstrates a clear advantage, yielding the highest median fitness among all mutation operators. These results suggest that PMX crossover combined with insertion mutation is likely the most effective combination for maintaining solution quality over time in this context.

Finally, the impact of the selection method was also compared, (Figura 10). Both tournament and ranking selection were evaluated, with tournament selection significantly outperforming ranking. This can be attributed to tournament selection's direct competition mechanism, which consistently favors stronger individuals while maintaining diversity through randomness. In contrast, ranking selection assigns selection probabilities based on fitness ranking, which may not exert enough selective pressure in highly competitive scenarios like lineup optimization.

The plot in Figura 11 shows the average fitness progression over generations for the best 5 performing configurations. The shaded areas represent on standard deviation above and below the mean, illustrating the variability of each configuration across multiple runs. Observing both the mean and variability helps us assess not only which configuration achieves higher fitness but also which ones are more consistent.

Looking at the plot, we can see that line blue (pmx=0.8; insertion= 0.2; tournament_elitism=False) outperforms the other configurations from generation 70 on. However, as we have seen on the isolation of impact of elitism that elitism set as True, improves our algorithm performance, acknowledging that we decided to keep the 3rd configuration (pmx=0.8, insertion= 0.2, tournament_elitism=True). With all the visualisation from the isolations of methods and some critical sense we arrived at our best set up which is **elitism=True, PMX, Insertion mutation and tournament selection**.

4.2 Fine-Tuning Operator Probabilities

After selecting the best operator types, we moved to the second phase, where we increased the population size and number of generations to 200. This allowed for a deeper search in the solution space and better convergence stability.

We tested different combinations of crossover and mutation probabilities. Specifically, we sampled 20 random pairs from a grid of 5 evenly spaced values between 0 and 1 for each probability, resulting in 25 possible combinations. Again, each was executed 30 times to ensure robust statistical evaluation.

From Figura 12 we can conclude that the best-performing configuration was Crossover probability: 0.91; Mutation probability: 0.50; Crossover method: PMX; Mutation method: Insertion; Selection method: Tournament; Elitism: Enabled.

Best average fitness: 0.71.

This configuration was used for a final manual run, that is, with all parameters defined, from which we extracted the optimal festival lineup (Figura 13)

5. CONCLUSION

This project successfully demonstrated the use of a genetic algorithm to solve a scheduling problem for a multi-stage music festival. By carefully balancing three main objectives of maximizing prime-time popularity, ensuring genre diversity, and minimizing artist conflicts, increasing overall satisfaction among the audience.

Through some testing, we evaluated multiple combinations of genetic operators, including selection mechanisms, crossover strategies, and mutation techniques. Our analysis revealed that PMX crossover, insertion mutation, and tournament selection with elitism outperformed other configurations in terms of convergence speed and final solution quality. Fine-tuning of operator probabilities (probability of xo = 0.91; probability of mutation = 0.50) further enhanced performance, closing in a final setup that achieved a fitness of 0.71.

However, there is room for improvement. One key area for future work is a deeper analysis of the impact of parameter settings within the selection methods. Varying parameters such as tournament size, selection pressure, or selection probabilities could significantly influence the convergence behavior and quality of the solutions.

Overall, the project highlights how computational intelligence, particularly evolutionary algorithms, can be effectively applied to real-world optimization problems involving large and complex search spaces. The final result offers a data-driven, optimized festival lineup that balances audience interest and scheduling practicality, demonstrating both technical and practical applicability.

6. APPENDIX

Figura 1 - Fitness Goals

Order Crossover (OX)

Parent 1	1	2	3	4	5	6	7	8
Parent 2	8	7	6	5	4	3	2	1
Child 1	7	6	3	4	5	2	1	8
Child 2	2	3	6	5	4	7	8	1

Figura 2 - Schema of Order Crossover (OX)

Partially Matched Crossover (PMX)

Parent 1	1	2	3	4	5	6	7	8
Parent 2	8	7	6	5	4	3	2	1
Child 1	8	7	3	4	5	6	2	1
Child 2	1	2	6	5	4	3	7	8

Figura 3 - Schema of Partially Matched Crossover (PMX)

Insertion Mutation

Original	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Mutated	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20	21	22	23	24	25	26	27	28	29	18	30	31	32	33	34	35

Figura 4 - Schema of Insertion Mutation

Prime Slot Mutation

Original	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Mutated	1	2	3	4	5	6	7	8	9	10	11	12	13	14	28	16	17	18	19	20	21	22	23	24	25	26	27	15	29	30	31	32	33	34	35

Figura 6 - Schema of Prime Slot Mutation

Slot Shuffle Mutation

Original	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Mutated	1	2	3	4	5	6	7	8	9	10	11	12	13	21	15	16	17	18	19	20	35	22	23	24	25	26	27	28	29	30	31	32	33	34	14

Figura 5 - Schema of Slot Shuffle Mutation

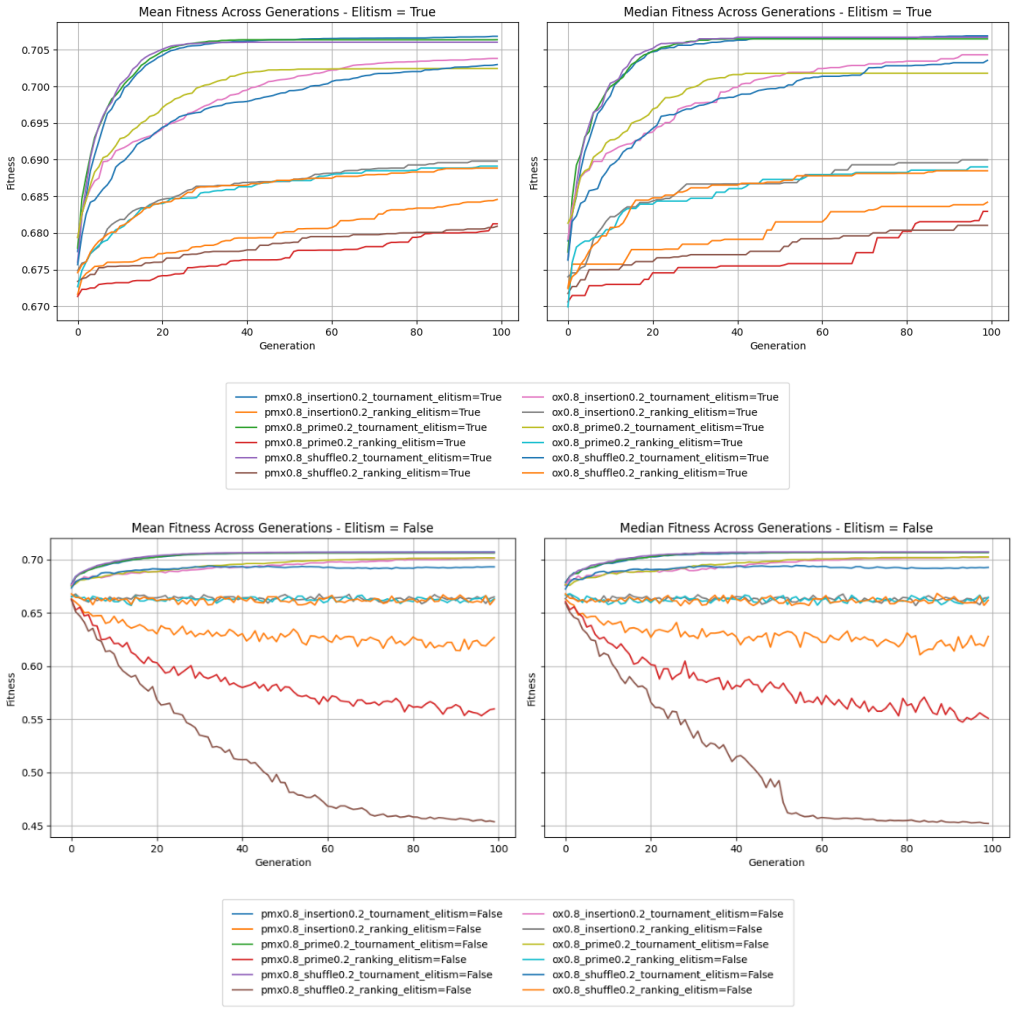

Figura 7 - Impact of Elitism on the fitness score

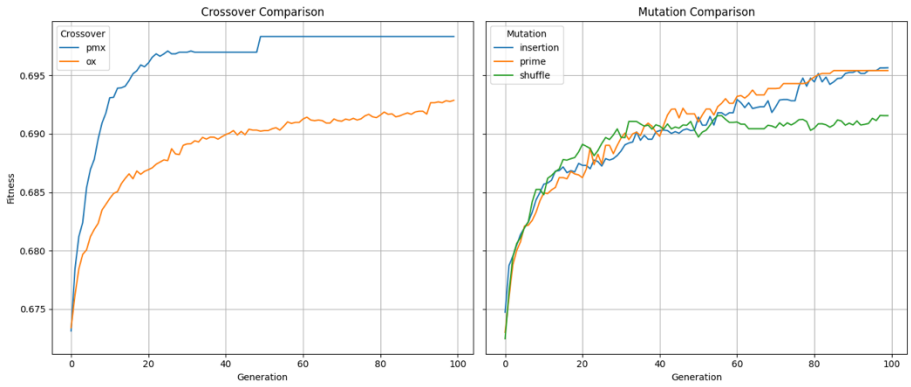

Figura 8 - Impact of Crossover and Mutation Operators

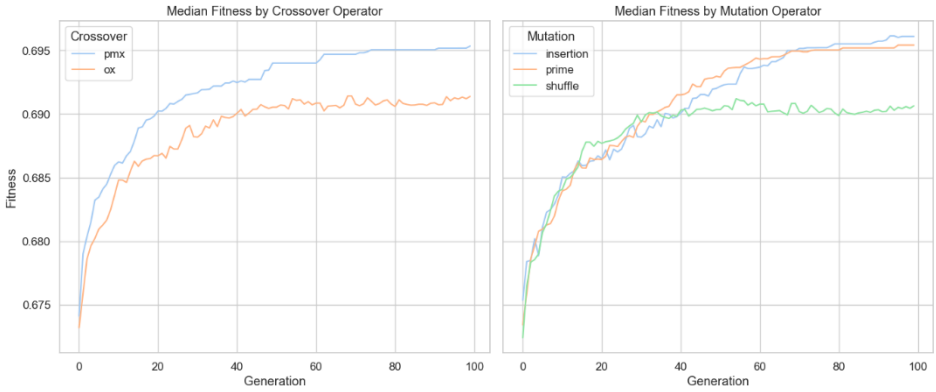

Figura 9 - Impact of Crossover and Mutation on the Median fitness score

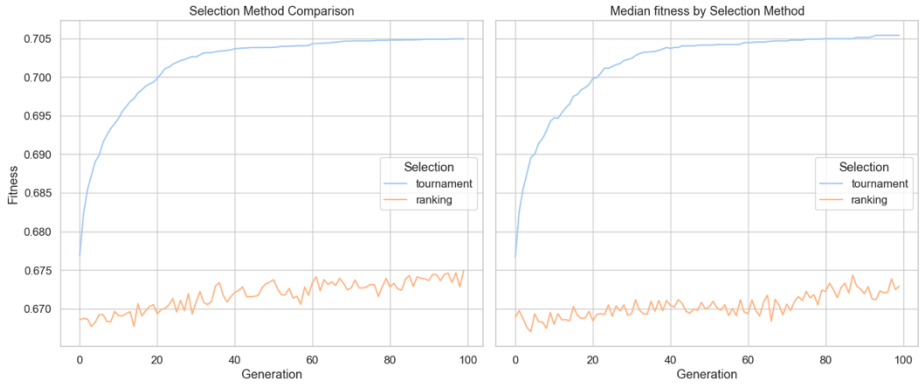

Figura 10 - Impact of Selection Method

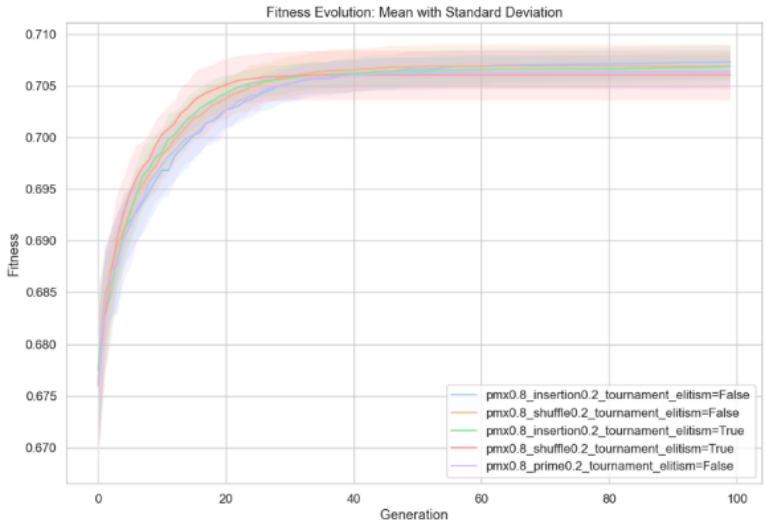

Figura 11 - Fitness evolution per generation

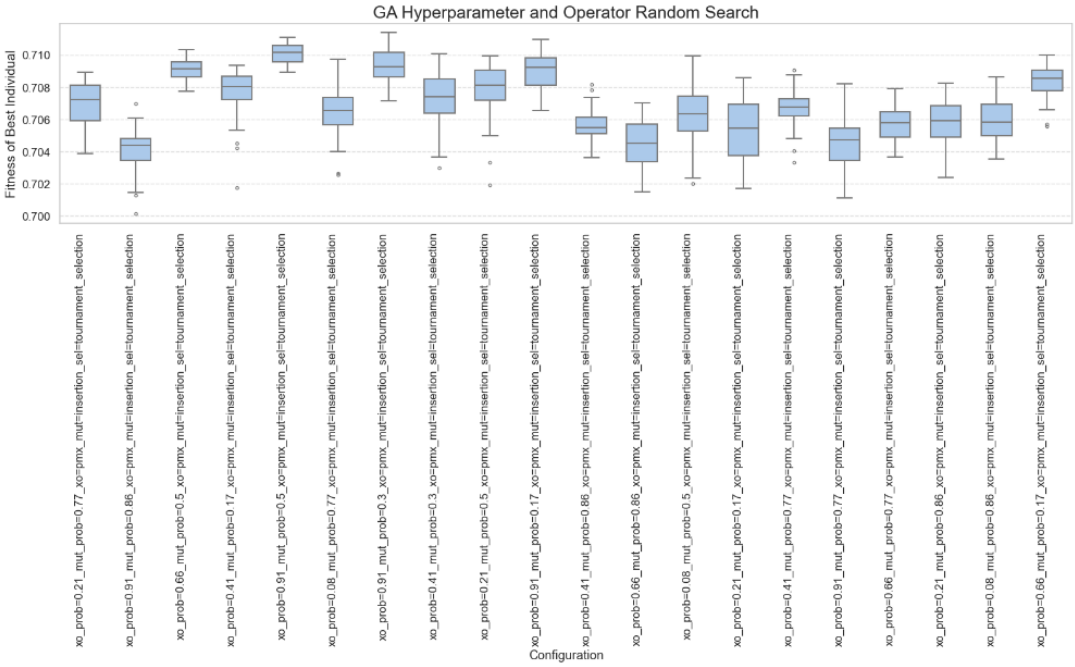

Figura 12 - GA Hyperparameter and Operator Random Search

	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7
Stage 1	Aurora Skies (75)	Crimson Harmony (20)	Deep Resonance (90)	Synthwave Saints (94)	Hypnotic Echoes (77)	Celestial Voyage (95)	Cloud Nine Collective (97)
Stage 2	The Wandering Notes (84)	Turbo Vortex (53)	The Sonic Drifters (88)	Nightfall Sonata (84)	Midnight Echo (75)	Mystic Rhythms (78)	Quantum Beat (96)
Stage 3	Solar Flare (78)	Parallel Dimension (58)	Astral Tide (69)	Phantom Groove (47)	The Bassline Architects (61)	Cosmic Frequency (53)	Lunar Spectrum (99)
Stage 4	Golden Ember (61)	Static Mirage (94)	Shadow Cadence (66)	Velvet Pulse (35)	Echo Chamber (98)	The Jazz Nomads (64)	Neon Reverie (100)
Stage 5	The Silver Owls (85)	The Polyrhythm Syndicate (66)	Harmonic Dissonance (96)	Blue Horizon (51)	Rhythm Alchemy (94)	Velvet Underground (72)	Electric Serpents (99)

Figura 13 - Final Schedule