Computational Intelligence for Optimization Project
Master's in Data Science and Advanced Analytics

NOVA Information Management School
Universidade Nova de Lisboa

Music Festival Lineup Optimization
Report

Group BB

Bruna Simões, 20240491

Dinis Pinto, 20240612

Marco Galão, 20201545

Margarida Cardoso, 20240493

Spring Semester 2024-2025

TABLE OF CONTENTS

1. Introduction

The full project is available on our GitHub repository [Bib. 1] for further exploration and collaboration.

To address the proposed optimization problem, an implementation of the Genetic Algorithm (GA) introduced in class was used and extended with four custom mutation operators, two crossover operators, and three distinct selection mechanisms, that were considered adequate to the problem's specific constraints and objectives. A comprehensive search was carried out to identify optimal configurations and parameters, followed by a performance comparison with alternative optimization neighborhood-based search algorithms. The final stage consisted of testing the hypothesis: what if diversity is introduced when the algorithm performance metrics (fitness) begin to stagnate?

2. Problem Definition

The Music Festival Lineup Optimization problem aims to allocate thirty-five artists across five equivalent stages and seven equivalent time slots, assembling a lineup that assigns exactly one artist to one stage and one slot, without leaving any artist unassigned or creating repetitions. An optimal solution maximizes the genre diversity of artists performing at the same time across different stages, avoids fan conflicts caused by overlapping audiences in simultaneous performances, and maximizes the popularity of the artists scheduled in the prime slot - the final time slot on each stage.

An individual in this optimization problem is represented by a 2D matrix, with five columns (the stages) and seven rows (the time slots). Each cell contains one artist, allowing for a clear and intuitive mapping of the full festival program.

The search space consists exclusively of valid permutations that assign all artists across the time slots without repetition and without any stage being left out - a total of $35!$ possible solutions.

The fitness function is computed in three modular components, which are added together to obtain the final fitness score:

- **Prime slot popularity score:** calculated by summing the popularity scores of the five artists scheduled in the prime slots of the current solution and dividing this sum by the theoretical maximum - the sum of the popularity scores of the five most popular artists in the dataset.
- **Genre diversity score**: computed for each time slot so that for each row (i.e., time slot across stages), the number of unique genres is counted and normalized by the theoretical maximum diversity - the minimum between the number of stages and the number of unique genres in the dataset. The final diversity score is the average across all time slots.
- **Conflict penalty:** consists of the sum of all pairwise conflict values between artists performing in the same time slot, then normalized by the maximum possible conflict, representing the worst-case scenario of scheduling the most conflicting artists together. Since this is a penalty, its value is subtracted in the final aggregation.

These three components are all bounded in the interval $[0,1]$ and are combined by summation to form the final fitness score, bounded between -1 (corresponds to the worst possible solution - maximum conflict, no diversity, and no popular artists in prime slots) and 2 (represents the ideal solution - maximum diversity, no conflicts, and most popular artists in prime slots). Such formulation guarantees that all components have an equal contribution to the final score, ensuring a well-balanced optimization over all objectives. The fitness function and selection methods were adapted to support negative fitness values, as this range was considered more informative for tracking evolutionary progress than normalizing the final fitness scores to the $[0, 1]$ interval.

3. Selection and Genetic Operators

3.1 Selection

To select the individuals in which to apply the genetic operators we decided to implement the three most common selection algorithms: fitness proportionate selection, the ranking selection and the tournament selection. [Bib. 2]

3.1.1 Fitness proportionate selection

The rationale behind this selection method is that the higher the fitness of an individual, the higher the probability of it being selected. Since the objective is to maximize fitness, the probability of an individual being selected is the proportion of its fitness compared to the sum of all individuals' fitness. This can be visualized as a roulette wheel split into several slices, each slice representing an individual. The size of the slice is proportional to the fitness of the individual and selection occurs by simulating a spin of the wheel [Fig. 1]. Being a fitness-based method can offer a key advantage: it naturally favors better-

performing individuals while still giving weaker ones a chance to be selected, which helps maintain diversity in the population during early generations. [Bib. 3]

3.1.2 Ranking selection

Our implementation begins by sorting the individuals in the population based on their fitness from best to worst. For each individual, the selection of it being selected is a function of the position it occupies in the ranking, so the higher the rank the higher the probability of it being selected. The linear function that determines the probability of an individual being selected is $P(sel\ i_k) = \frac{N - position_{i_k} + 1}{\sum_{j=1}^{N}(N - position_{i_j} + 1)}$ [Fig. 2]. This formulation ensures that higher-ranked individuals are still favored but it avoids an while over-reliance on absolute fitness values, which helps maintain a controlled selection pressure [Bib 4].

3.1.2 Tournament selection

This selection method is the least computationally expensive, as it only evaluates the fitness of N individuals randomly selected from the population using a uniform distribution. The selected individual is the one with the highest fitness among the N sampled candidates. The constant N is known as the tournament size, and it determines the strength of the selection pressure — larger values of N increase the likelihood of selecting fitter individuals, while smaller values help preserve diversity in the population [Fig. 3]. We opted to set this parameter to 4 when running the grid search for selection and genetic operators. This choice is supported by Goldberg and Deb (1991), that showed that a tournament size in the range of 2–5 offers a good balance between selection pressure and population diversity. [Bib 5]

3.2 Crossover

Two distinct crossover operators were implemented to solve this problem. The operators differ in how they combine information from the parents - one following a more classical permutation approach and the other focusing on local performance by considering the fitness of each slot. Both ensure that each artist appears exactly once in the offspring.

3.2.1 Partially Mapped Crossover

This crossover aims to preserve relative order and structure from both parents. This operator randomly selects two crossover points, defining a central window of consecutive elements to be inherited directly from one parent. The remaining positions are then filled using elements from the other parent, but whenever a conflict arises (an artist already present in the window), a mapping is used to trace the corresponding replacement based on the values swapped in the window, avoiding repeated artists. In this implementation, individuals are first flattened into a one-dimensional sequence and once the full offspring are constructed, it is reshaped back to its original matrix form. Refer to Figure 4 for a visual representation.

3.2.2 Fitness Based Slot Crossover

This crossover selects the best-performing slots from both parents based on a local fitness score that combines genre diversity and conflict penalty. From the 14 available slots (7 from each parent), the 7 highest ranked are selected to build a single offspring, which may initially contain repeated artists. To ensure a valid permutation, the duplicates are replaced with randomly chosen missing artists, starting from the weakest slots to preserve the strongest ones. This operator returns only one child, as the selection process is greedy and would produce highly similar offspring if repeated. [Fig. 5]

3.3 Mutation

In all implemented mutation operators, modifications were only performed between different rows (i.e., between different slots). This decision was based on the observation that, in the context of this problem, changing the position of artists within the same row (i.e., changing the stage in which they perform during a given time slot) does not affect the fitness of a solution. Therefore, to ensure that mutations have a meaningful impact on solution quality, intra-row changes were avoided.

3.3.1 N Swap Mutation

The N Swap Mutation works by randomly selecting pairs of elements in the solution matrix and swapping them, ensuring that each swap involves elements from different rows [Fig. 6]. A total of n swaps is performed. For each swap, two distinct positions in the matrix are randomly selected. If both positions lie within the same row, they are discarded, and new positions are drawn until the elements belong to different rows. Once a valid pair is identified, the elements at these positions are exchanged. This process is repeated n times, ensuring that each swap introduces a small structural change

while preserving the general integrity of the solution. The randomness introduced by the mutation helps maintain genetic diversity in the population and allows exploration of different arrangements across time slots.

3.3.2 Scramble Mutation

The Scramble Mutation works by selecting a random segment within a single column of the solution matrix and randomly shuffling its elements [Fig. 7]. The process begins by randomly choosing a column, followed by the selection of a segment length, constrained by a predefined maximum. Once the segment is identified, its elements are shuffled randomly to produce a new ordering. This shuffled segment is then reinserted into the matrix, replacing the original one. This localized rearrangement introduces variation without altering the overall content of the column, helping to maintain diversity in the population and enabling the exploration of alternative scheduling configurations within individual stages.

3.3.3 Prime Slot Swap Mutation

To better explore the *prime slot popularity* objective, the Prime Slot Swap Mutation was implemented. This mutation targets the last row of the solution matrix, which corresponds to the prime-time slot. It randomly selects one of the remaining time slots and swaps it with the prime slot [Fig. 8]. By doing so, it shifts entire line-ups into or out of the prime-time position, allowing the algorithm to test different artist placements in the most desirable slot. This mechanism promotes the discovery of solutions that more effectively align popular artists with the prime-time slot, while preserving the composition of each individual time slot.

3.3.4 Preserve Best Slots Mutation

To further exploit partial structures that already perform well, the Preserve Best Slots Mutation focuses on maintaining high-quality time slots while reassigning the rest. Each row is evaluated individually based on its fitness - considering only the *genre diversity* and *conflict penalty* objectives, as the *prime slot popularity* cannot be computed at the time slot level. A fraction of the top-performing slots, determined by a predefined ratio, is preserved unchanged. The elements from the remaining slots are gathered into a pool and shuffled before being reassigned to those slots. This mutation is the most computationally expensive among those implemented, as it requires computing the fitness of every slot. It is also the most disruptive, since it can reallocate a large portion of the solution's content in a single step. Nevertheless, it balances exploitation and exploration by preserving promising substructures while aggressively reshaping the rest of the solution space. [Fig. 9]

4. Performance Analysis

With the foundational classes and functions implemented and tested, the next step was to evaluate and compare the performance of the optimization algorithms. According to the No Free Lunch Theorem, no single algorithm is optimal for all problems, making algorithm comparison a crucial part of the project.

All the implemented algorithms are stochastic, meaning their results can vary between runs. To ensure statistical reliability, each configuration was executed 30 times, as this is generally the minimum number of runs recommended for robust performance comparisons. The performance of each configuration was assessed using:

- **Statistical metrics:** mean and median fitness (more robust to outlier runs) and standard deviation (to evaluate stability).
- **Visual analysis:** line plots, boxplots, heatmaps, and network graphs.
- **Statistical significance testing:** the Wilcoxon signed-rank test was used to determine whether differences between algorithm performances were statistically significant.

While efforts were made to ensure a fair comparison (e.g., consistent evaluation conditions), some bias is inevitable due to the limited exploration of the full parameter space. The performance analysis was carried out in four stages, described in the following sections.

4.1 Genetic Algorithm - Configuration Comparison

The first step was to evaluate the full range of GA configurations by testing all possible combinations of selection methods, crossover operators, and mutation operators (24 combinations). To ensure a fair comparison, all configurations were run using the same baseline parameters, chosen to balance optimization performance with computational cost, as detailed in Table 1 and 2. Configurations without elitism were excluded, as elitism is known to enhance convergence stability and

solution quality. This allowed a more focused exploration of operator combinations while keeping elitism consistently enabled.

To compare the results, line charts illustrating the evolution of mean and median across 30 runs per generation were used. From these plots [Fig. 10], two configurations emerged as the most promising: (1) tournament selection with partially mapped crossover and N swap mutation, achieving a median final fitness of 1.5880, and (2) tournament selection with partially mapped crossover and scramble mutation, with a median final fitness of 1.5717. The first configuration showed superior performance from around the 35th generation onward.

Although tournament selection includes configurations that tend to converge more prematurely, it also produced the best performing setup. Ranking selection performed reasonably well, particularly in early generations. Fitness-proportionate selection resulted in slower and more gradual convergence, likely due to its weaker selective pressure and more uniform sampling across individuals, which limited its ability to consistently guide the search toward better solutions.

Complementing this, the boxplots of final-generation fitness across the 30 runs [Fig. 11, 12] confirmed that, overall, configurations using tournament selection outperformed those using fitness-proportionate selection, which in turn outperformed ranking selection. Moreover, the top-performing configuration also demonstrated strong stability, offering a consistent balance of high performance and reliability across runs.

An additional analysis isolated the impact of each GA operator [Fig. 13] by grouping all runs that used a particular operator and computing the median best fitness across those runs. This revealed that tournament selection was the most effective selection method, fitness-based slot crossover performed best among crossover operators, and N swap and scramble mutations were both top performers with very similar results. However, it is worth noting that while fitness-based slot crossover performed well on its own, the best overall configuration emerged from a combination that included partially mapped crossover.

Finally, a crucial statistical analysis was performed to visually and quantitatively assess the similarity between configurations, based on whether their performance differences were statistically significant. Using the Wilcoxon signed-rank test, a graph-based visualization [Fig. 14] was created where nodes represent configurations and distances reflect statistical dissimilarity. In this graph, the two leading configurations were shown to be significantly different, with a p-value of 0.017 (< 0.05) [Fig. 15].

Based on these comprehensive analyses, the GA configuration combining tournament selection, partially mapped crossover, and N swap mutation was selected as the best. This configuration likely performed best due to a balance achieved between its components, since tournament selection maintains selection pressure while preserving diversity [Bib. 5], partially mapped crossover recombines solutions preserving meaningful substructures [Bib. 6] and N-swap mutation introduces controlled and non-destructive variation allowing fine-tuning.

4.2 Best Genetic Algorithm Configuration vs. Hill Climbing and Simulated Annealing

Once the best GA configuration was identified (tournament selection, partially mapped crossover, and N swap mutation), it was compared against two other optimization methods:

- **Hill Climbing (HC)**, using the N swap mutation as its neighborhood function. It generated 5 neighbors per iteration, with little performance gain observed when increasing this number.
- **Simulated Annealing (SA)** configured as detailed in Table . It used the same N swap mutation as a neighborhood function but generated only a single neighbor per iteration.

These settings were selected based on intuition and to ensure a fair comparison - matching the number of iterations with the number of GA generations and using the same mutation operator as the neighborhood function.

The boxplot [Fig. 16], which summarizes the best fitness values found over 30 runs for each method, highlights clear performance differences. HC outperformed SA in terms of fitness (with a median best fitness of 1.2620 and 1.0105, respectively), with both showing similar levels of stability. However, GA clearly stood out as the superior approach, consistently achieving significantly higher fitness scores (with a median final fitness of 1.5880) with lower variability across runs - indicating both strong performance and robust convergence. This is consistent with literature and empirical experience that greedy, neighborhood-based heuristics, while efficient on simpler landscapes, underperform when faced with complex or highly constrained problems. Although SA allows for occasional acceptance of worse solutions to escape local optima, the single-solution approach limits the exploration capabilities, while GAs work with a highly diverse

population of candidate solutions and by using crossover and mutation they explore the space more thoroughly, allowing for better solutions.

As anticipated, the Wilcoxon signed-rank test confirmed these observations, with all p-values [Fig. 17] extremely close to zero - strongly reinforcing the statistically significant differences between the algorithms.

4.3 Hyperparameter Tuning of GA

Finally, the best-performing GA configuration was refined through hyperparameter tuning using random search (25 combinations), aiming to further boost performance. This approach randomly sampled values from ranges deemed reasonable based on intuition and preliminary experimentation, presented in Table 4.

To compare results, the same methodology from Section 4.1 was applied. A line chart tracking the evolution of mean and median fitness over 30 runs [Fig. 18] highlighted two standout configurations:

- **(1)** tournament selection (size = 2), partially mapped crossover (probability = 0.84), and N swap mutation (n = 2, probability = 0.36), achieving a median final fitness of 1.6311;
- **(2)** tournament selection (size = 4), partially mapped crossover (probability = 0.71), and N swap mutation (n = 2, probability = 0.39), with a median final fitness of 1.6285.

While the second configuration generally led in performance, it was slightly outperformed by the first in the final few generations.

A boxplot of final-generation fitness [Fig. 19] further supported the strength of these two configurations, showing similar levels of stability. The first configuration displayed a slight skew toward higher fitness values, suggesting it may offer a better balance of peak performance and consistency across runs.

To support decision-making, a statistical analysis was conducted using the Wilcoxon signed-rank test. The test [Fig. 20, 21] revealed no statistically significant difference between the two leading configurations, with a p-value of 0.360 (> 0.05).

Given these comprehensive analyses, neither configuration showed a clear statistical advantage over the other. However, to move forward with a definitive choice, configuration (1) was selected. This decision was guided by its slightly stronger performance in the final stages of evolution.

4.4 Adaptive Pressure and Controlled Diversity in GA Optimization

In this section, we implemented adaptive strategies to mitigate premature convergence and promote continued exploration during stagnation phases of the GA. Specifically, we defined a threshold for the standard deviation of the best fitness values across recent generations. When the standard deviation fell below this threshold, we triggered the following adaptive pressure mechanisms simultaneously:

- Incremented the number of swaps in the N-swap mutation.
- Decreased the tournament size by one unit to increase the selection pressure.

Multiple thresholds for standard deviation were tested to assess the sensitivity of the algorithm to different plateau detection criteria, ranging from 0.001 to 0.2.

In addition, we applied a partial replacement strategy that involves replacing 50% of the weakest individuals, thereby injecting diversity and reducing the risk of being trapped in local optima.

The results [Fig. 22, 23] suggest that aggressive adjustments to selection pressure or mutation intensity in response to fitness plateau do not result in consistent improvements. Introducing random diversity through partial replacement of weaker individuals appears slightly more effective, though the gains remain limited, with a median final fitness of 1.6447. Given its relatively better performance, this strategy was selected to generate the final solution.

5. Final Solution

After identifying the best-performing algorithm and its optimal configuration - GA with tournament selection (tournament size = 2), partially mapped crossover (probability = 0.84), and N swap mutation (n = 2, probability = 0.36) with partial replacement when fitness score plateau - it was executed once to simulate practical deployment. This run produced a solution [Fig. 24] with a fitness score of 1.6608, representing a strong and well-optimized outcome based on the defined objectives.

6. CONCLUSION

This project addressed a sophisticated permutation-based optimization problem through the development and empirical evaluation of a GA tailored to the Musical Festival Lineup. Multiple genetic operators were introduced– four mutation strategies, two crossovers and three selection methods – designed to balance exploration and exploitation within a highly constrained search space. Empirical results supported by statistical and visual analysis stated that the most effective configuration combined tournament selection, partially mapped crossover and N swap mutation. Not only did this configuration outperform HC and SA, but also showed great stability across runs. Further refinement through random hyperparameter search led to additional gains in convergence and fitness quality.

To prevent premature convergence, different variations of an adaptive strategy triggered by fitness stagnation were explored. Among these, partial replacement of weaker individuals proved to be the most effective, offering marginal improvements in final solutional quality. The final solution reflects a well optimized balance between genre diversity, conflict minimization and popularity maximization, highlighting the suitability of evolutionary algorithms to real-world problems.

Future extensions could focus on expanding the random search individually for each configuration of crossover, mutation and selection operators, optimizing the parameters for each configuration and testing additional operators and selection mechanisms to further assess the algorithm's robustness and adaptability. Finally, extending the problem formulation to include multi-objective optimization could more comprehensively capture trade-offs between prime slot popularity, genre diversity, and conflicts penalty.

ANNEX A: VISUAL REPRESENTATIONS

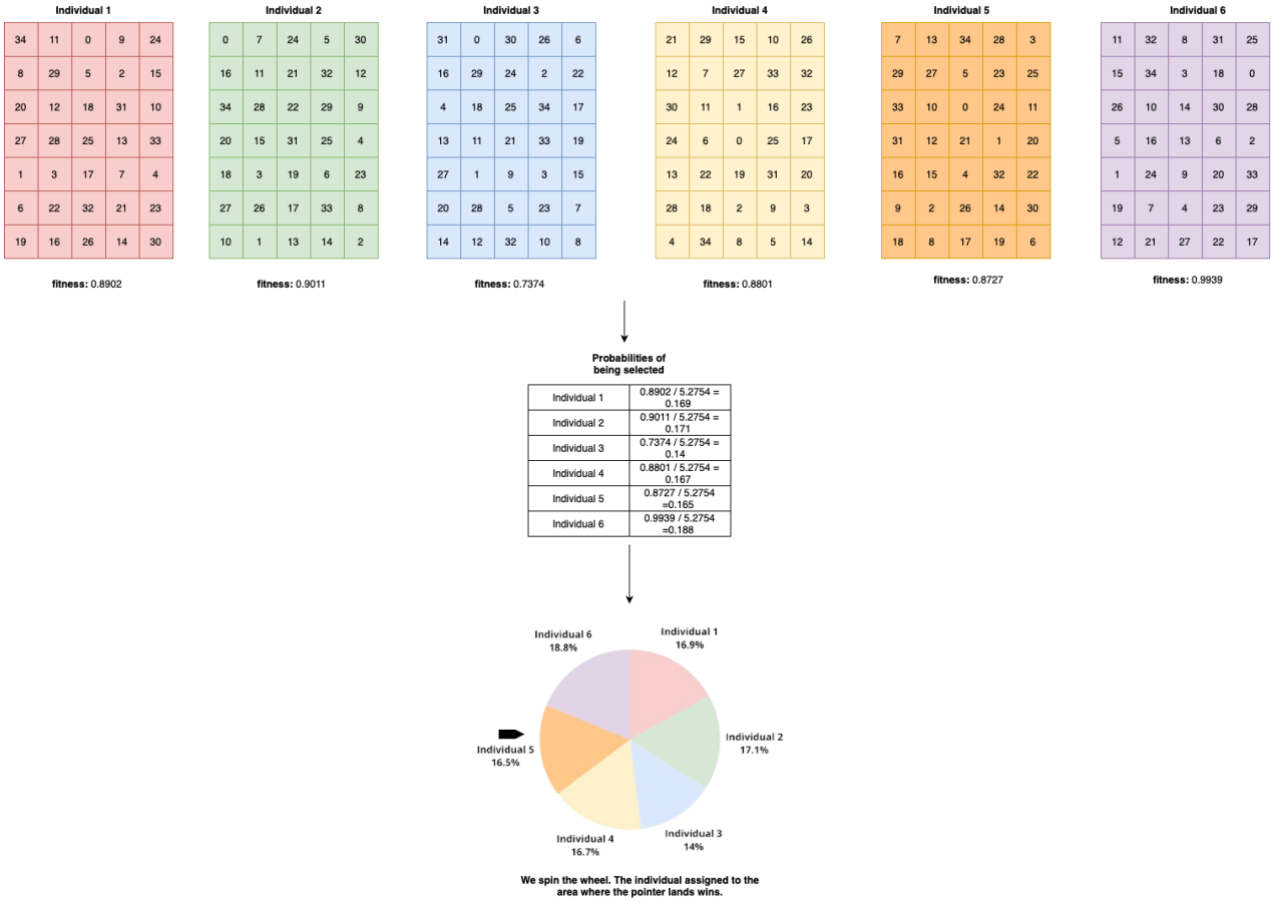


Figure 1 – Fitness Proportionate Selection

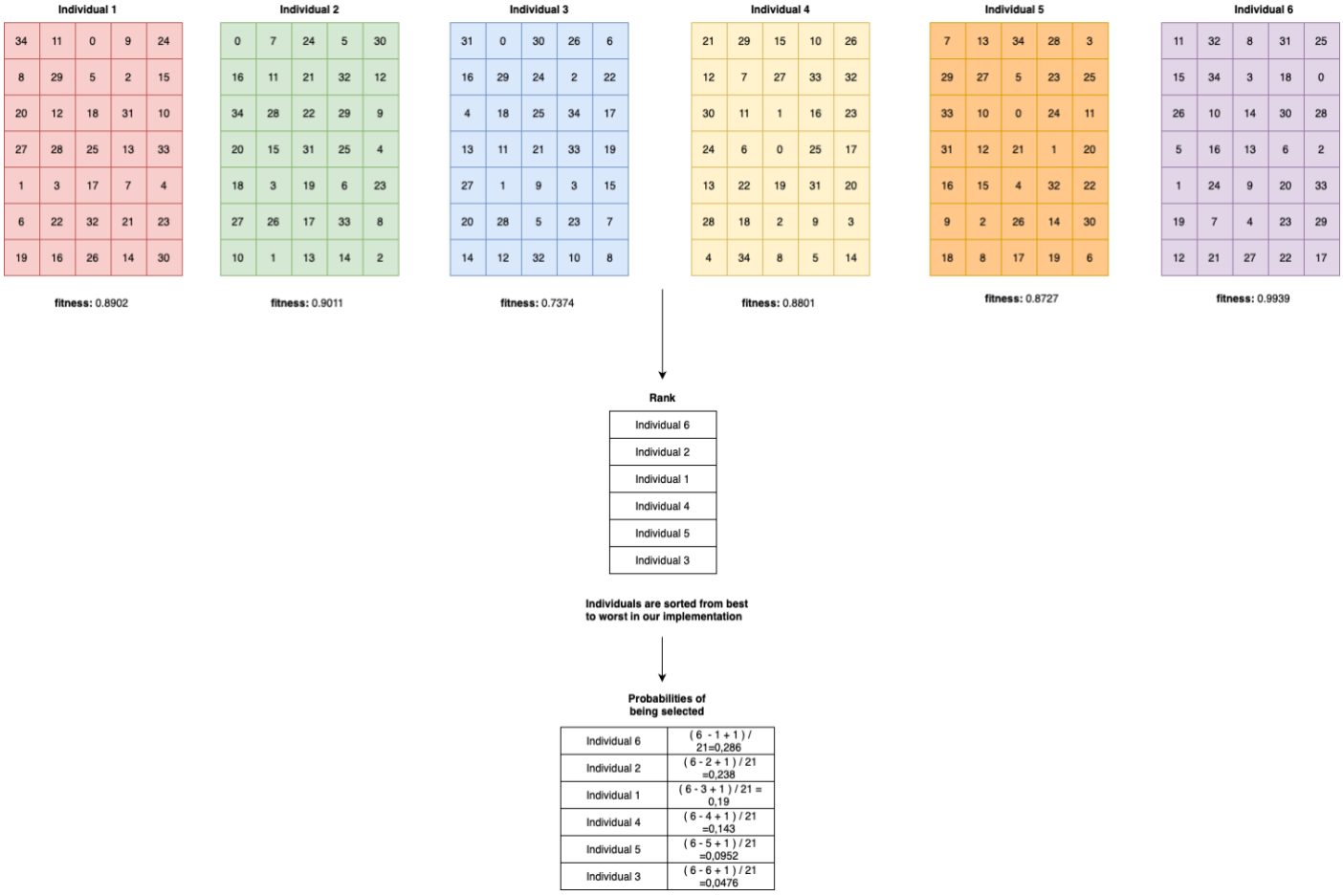


Figure 2 - Ranking Selection

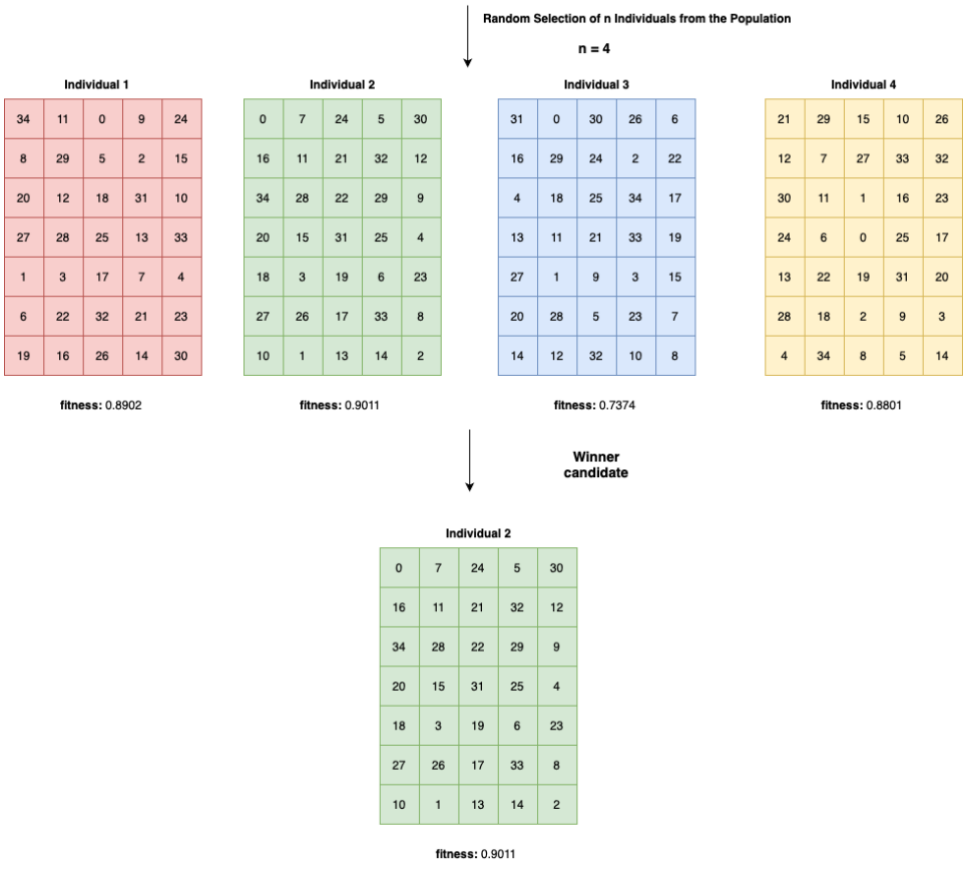


Figure 3 - Tournament Selection

Parent 1

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

Parent 2

19	8	3	10	16
7	34	20	18	11
0	15	6	13	22
32	27	30	5	21
2	14	33	23	1
31	17	29	4	9
12	24	25	26	28

Random Window: cx1=18, cx2=26

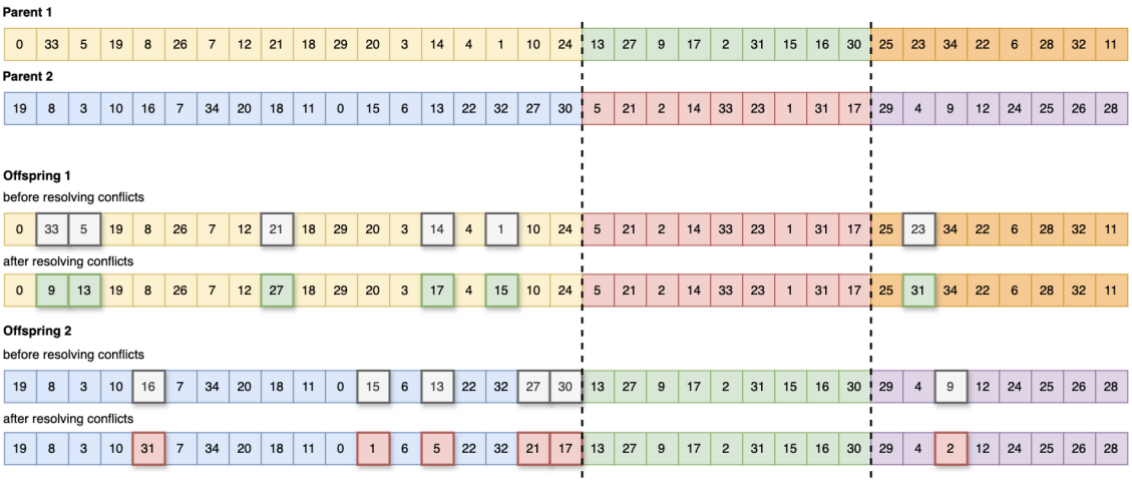


Figure 4 – Partially Mapped Crossover

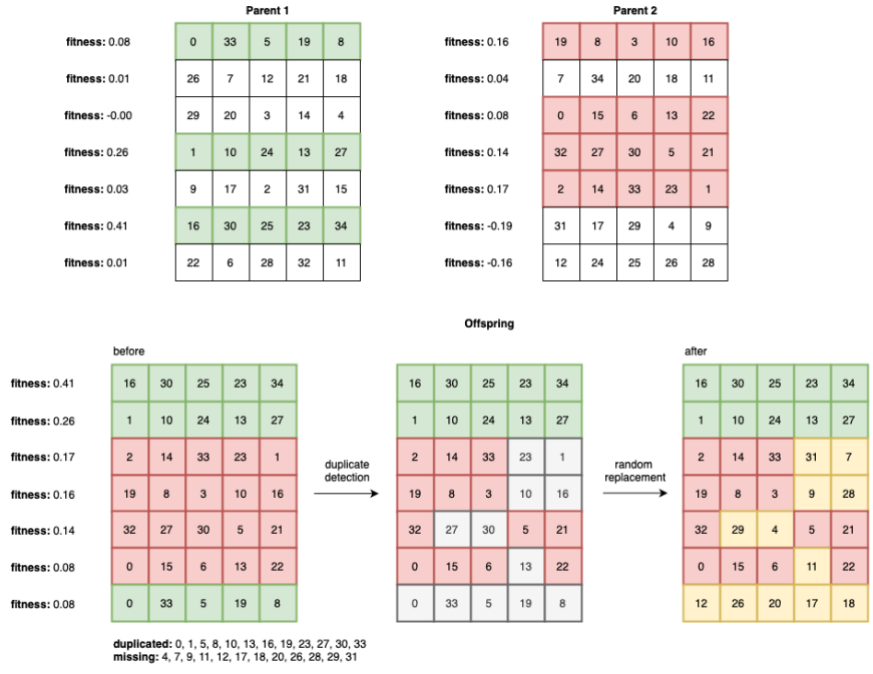


Figure 5 - Fitness Based Slot Crossover

Individual

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

Random Swap 1 - (row 0, col 2) <---> (row 2, col 0)

before

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

after

0	33	29	19	8
26	7	12	21	18
5	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

Random Swap 2 - (row 3, col 1) <---> (row 6, col 3)

before

0	33	29	19	8
26	7	12	21	18
5	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

after

0	33	29	19	8
26	7	12	21	18
5	20	3	14	4
1	32	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	10	11

Figure 6 - N Swap Mutation

Individual

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

Random Segment - col 3, segment length 3, rows 2-5

before

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

after

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	31	27
9	17	2	23	15
16	30	25	13	34
22	6	28	32	11

Figure 7 – Scramble Mutation

Individual

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

Random Row - row 1

before

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

after

0	33	5	19	8
22	6	28	32	11
29	20	3	14	4
1	10	24	31	27
9	17	2	23	15
16	30	25	13	34
26	7	12	21	18

Figure 8 – Prime Slot Swap Mutation

Individual

0	33	5	19	8
26	7	12	21	18
29	20	3	14	4
1	10	24	13	27
9	17	2	31	15
16	30	25	23	34
22	6	28	32	11

Keep Rows Ratio - best 50% rows

	before					after				
fitness: 0.08	0	33	5	19	8	0	33	5	19	8
fitness: 0.01	26	7	12	21	18	3	20	26	6	31
fitness: -0.00	29	20	3	14	4	9	4	17	28	32
fitness: 0.26	1	10	24	13	27	1	10	24	13	27
fitness: 0.03	9	17	2	31	15	29	12	21	7	11
fitness: 0.41	16	30	25	23	34	16	30	25	23	34
fitness: 0.01	22	6	28	32	11	2	22	14	18	15

Figure 9 - Preserve Best Slots Mutation

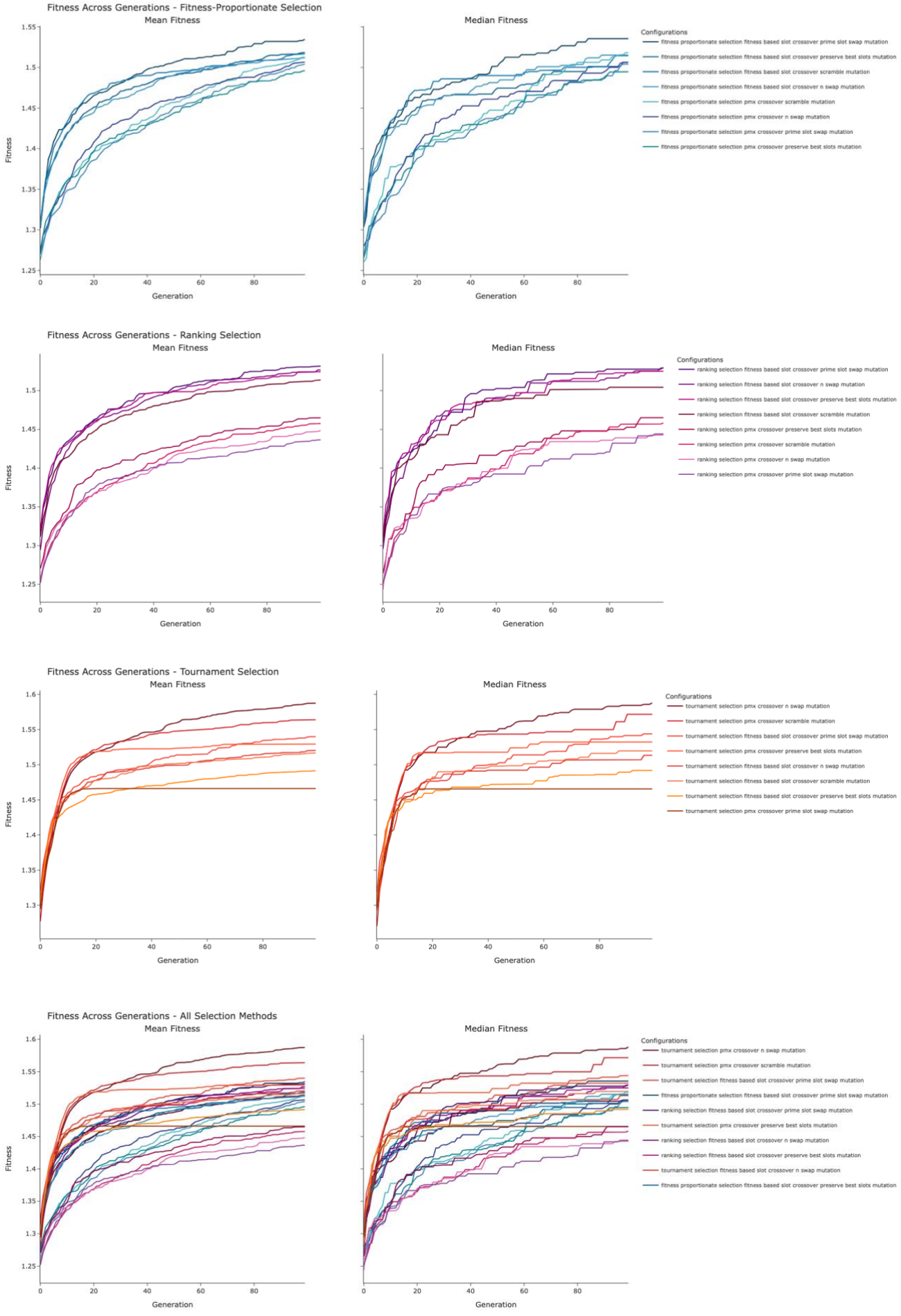


Figure 10 - Fitness Across Generations within Selection Methods

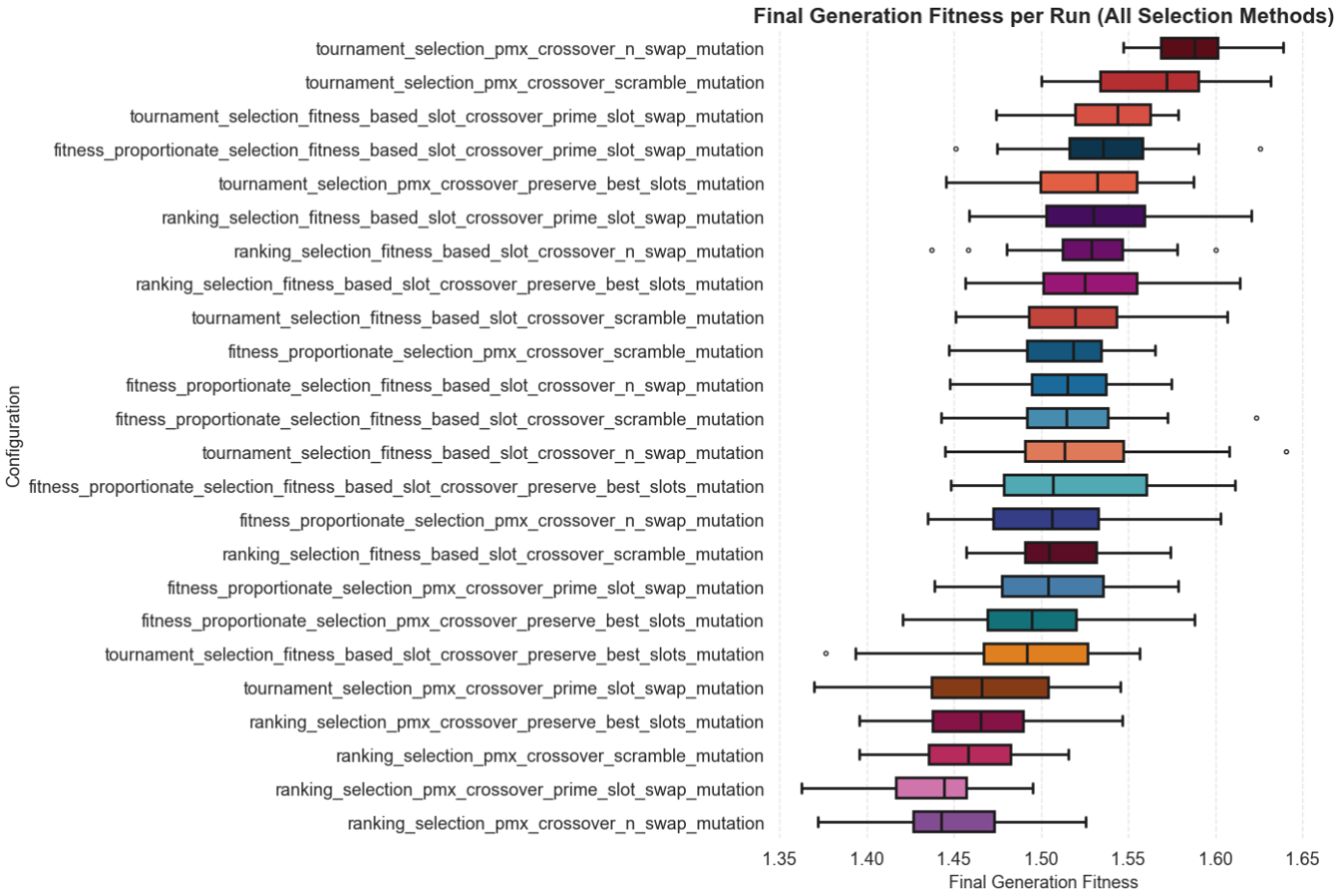


Figure 11 - Final Generation Fitness per Run within Combinations

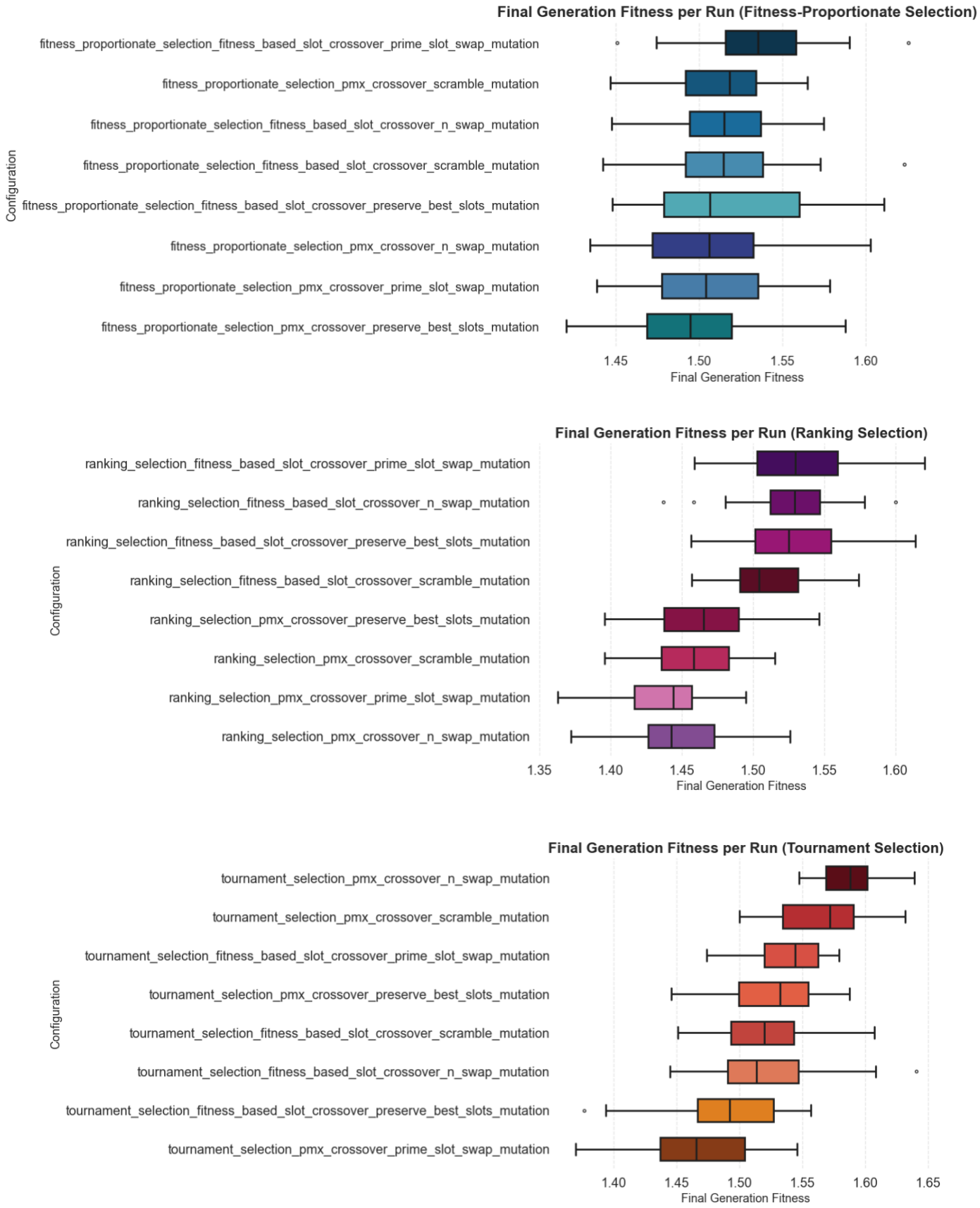


Figure 12 – Variance of Final Generation Fitness within Combinations

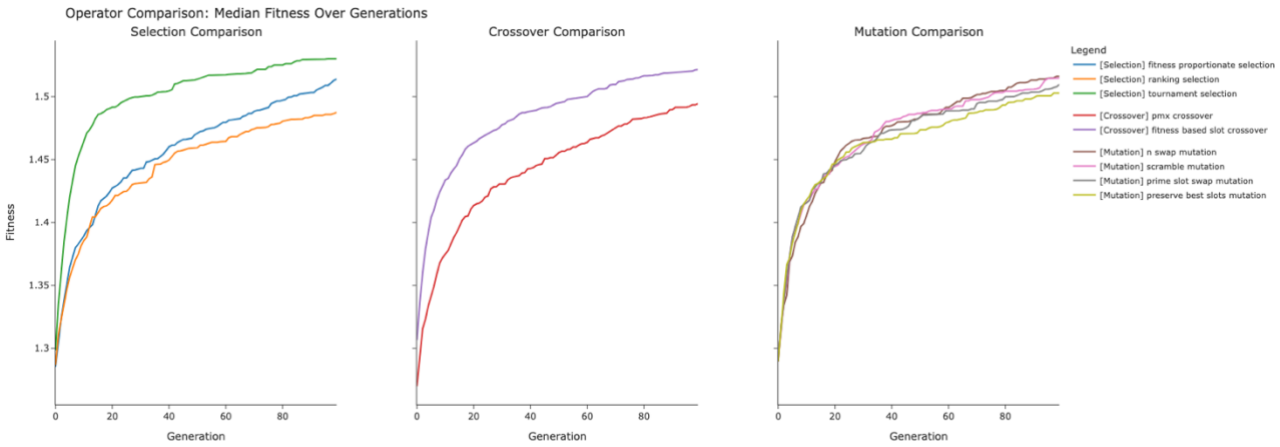


Figure 13 - Operator's Isolated Impact

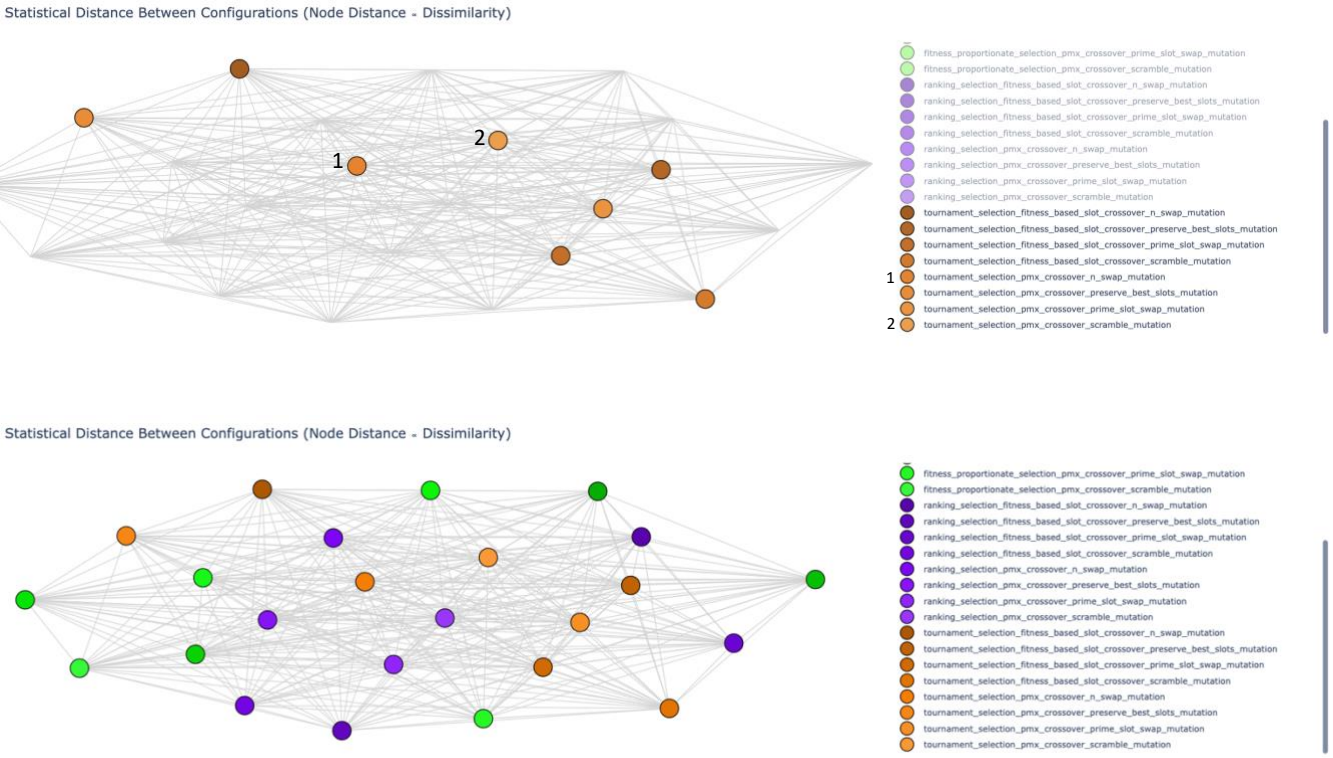


Figure 14 - Network Graph - Statistical Significance Representation

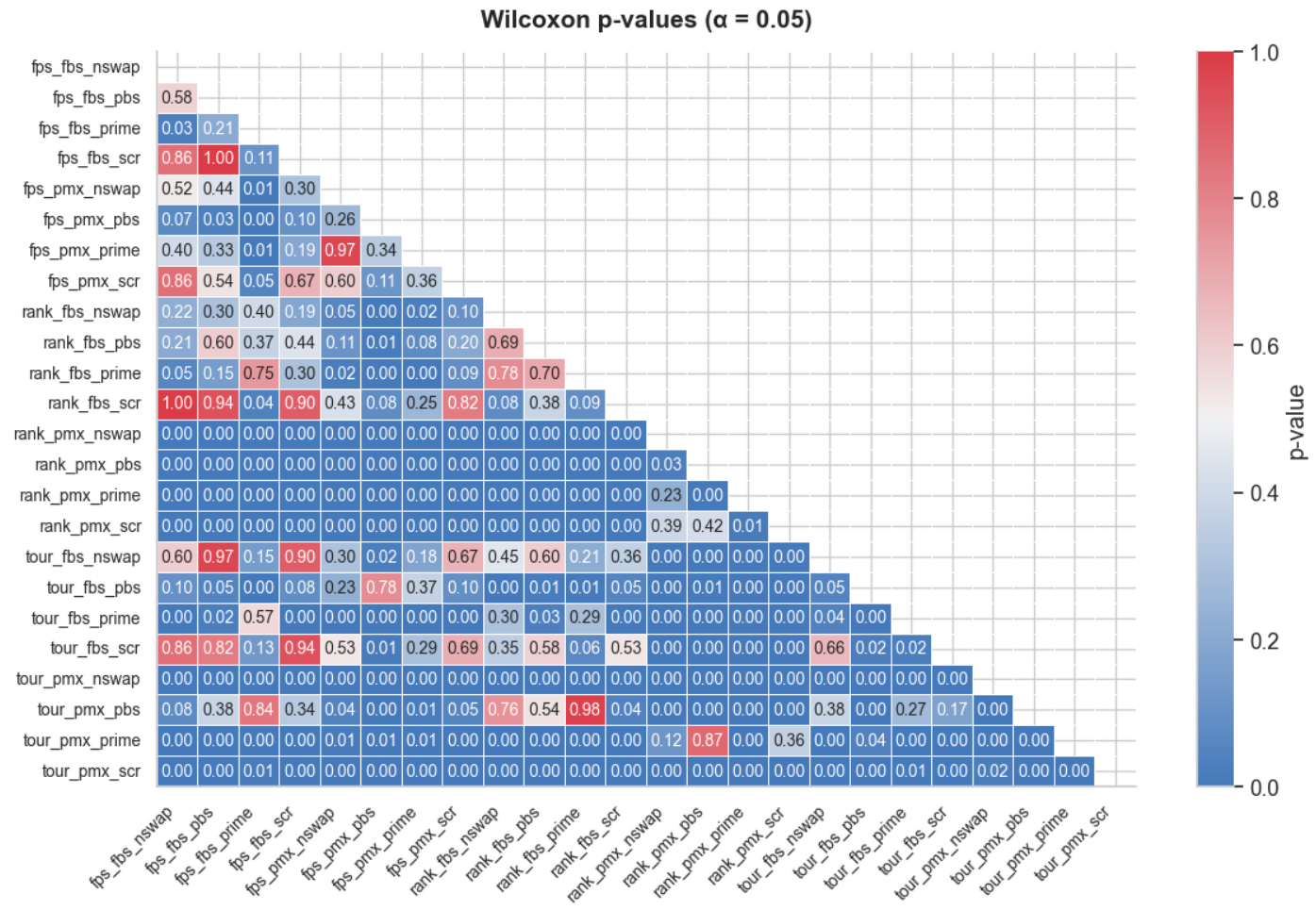


Figure 15 – Wilcoxon Signed-Rank Test p-values for GA Configuration Comparison

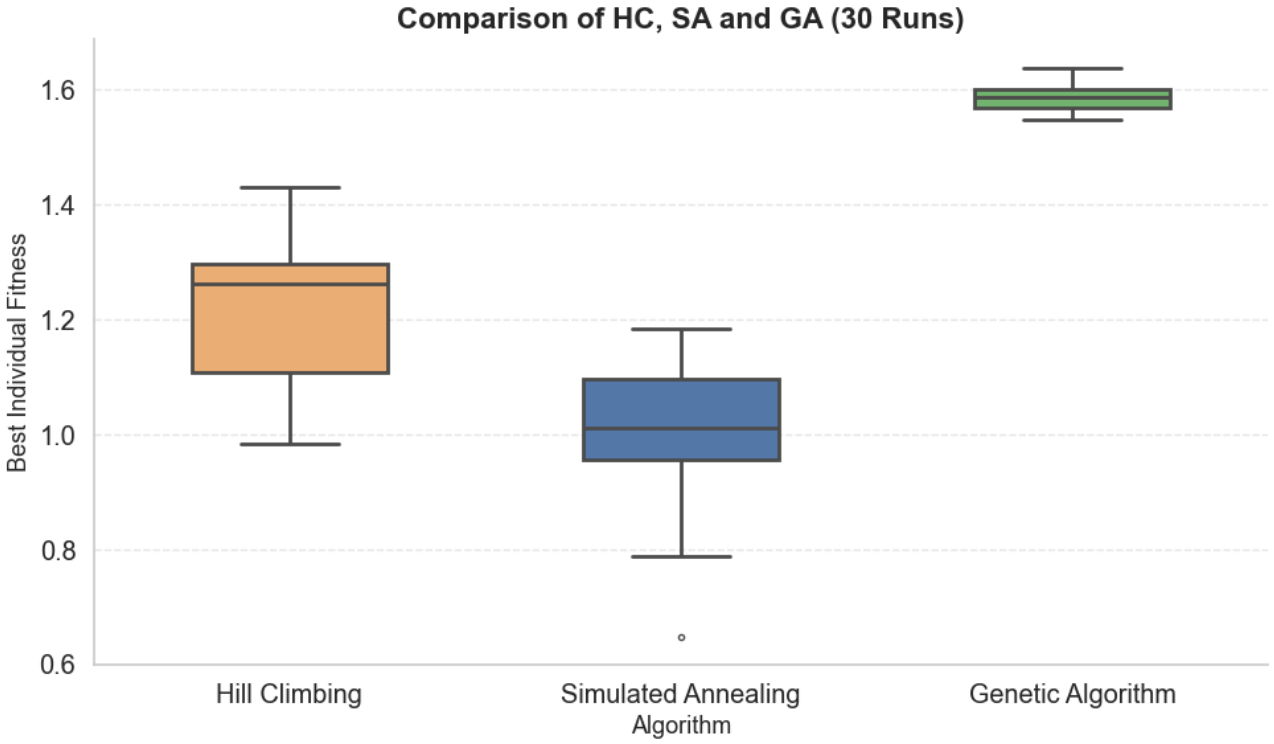


Figure 16 - Algorithms Best Individual Fitness Variance Comparison

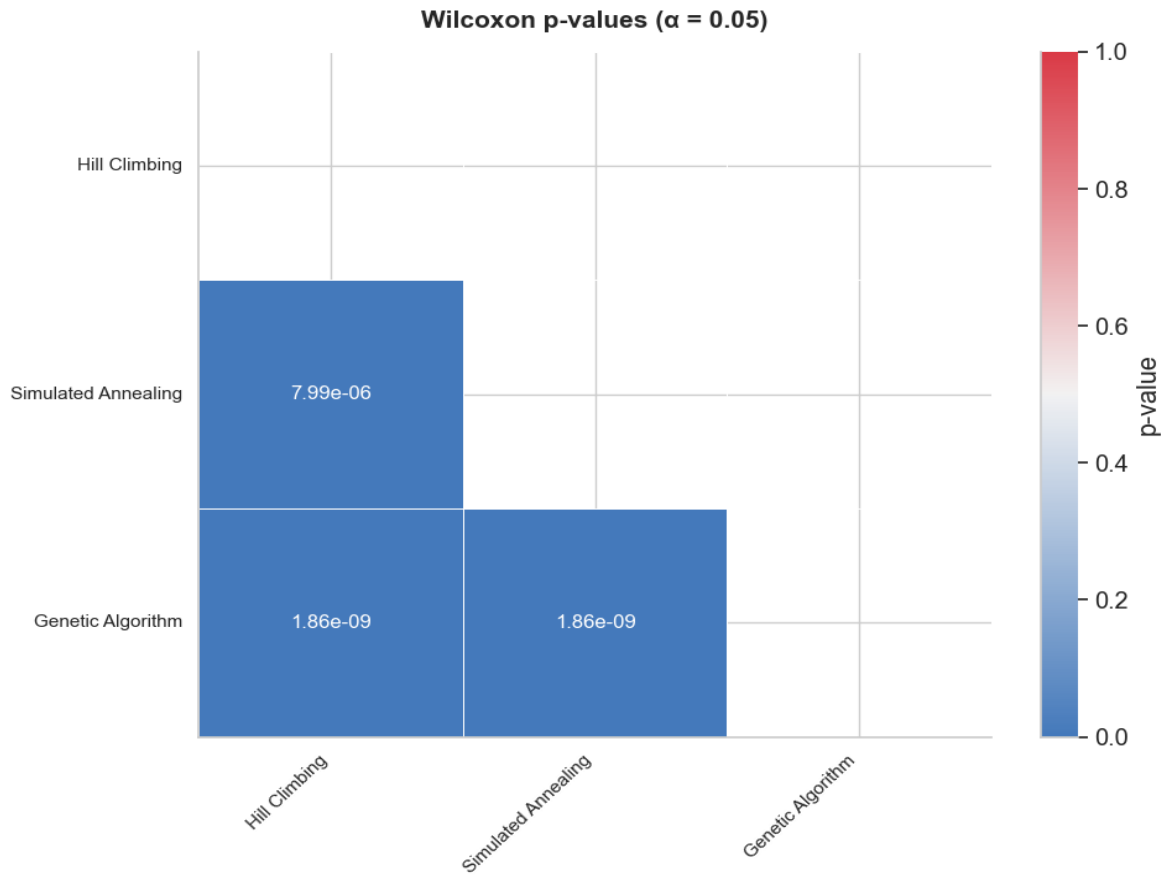


Figure 17 – Wilcoxon Signed-Rank Test p-values for HC, SA, GA Comparison

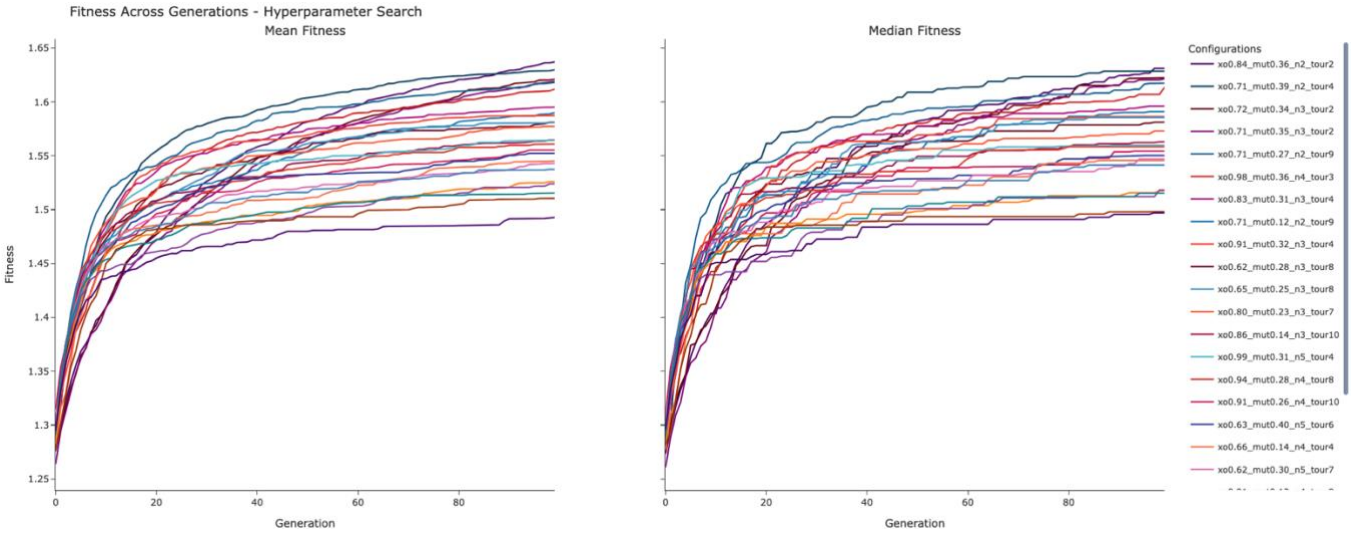


Figure 18 – Fitness Across Generations for All Hyperparameter Search Configurations

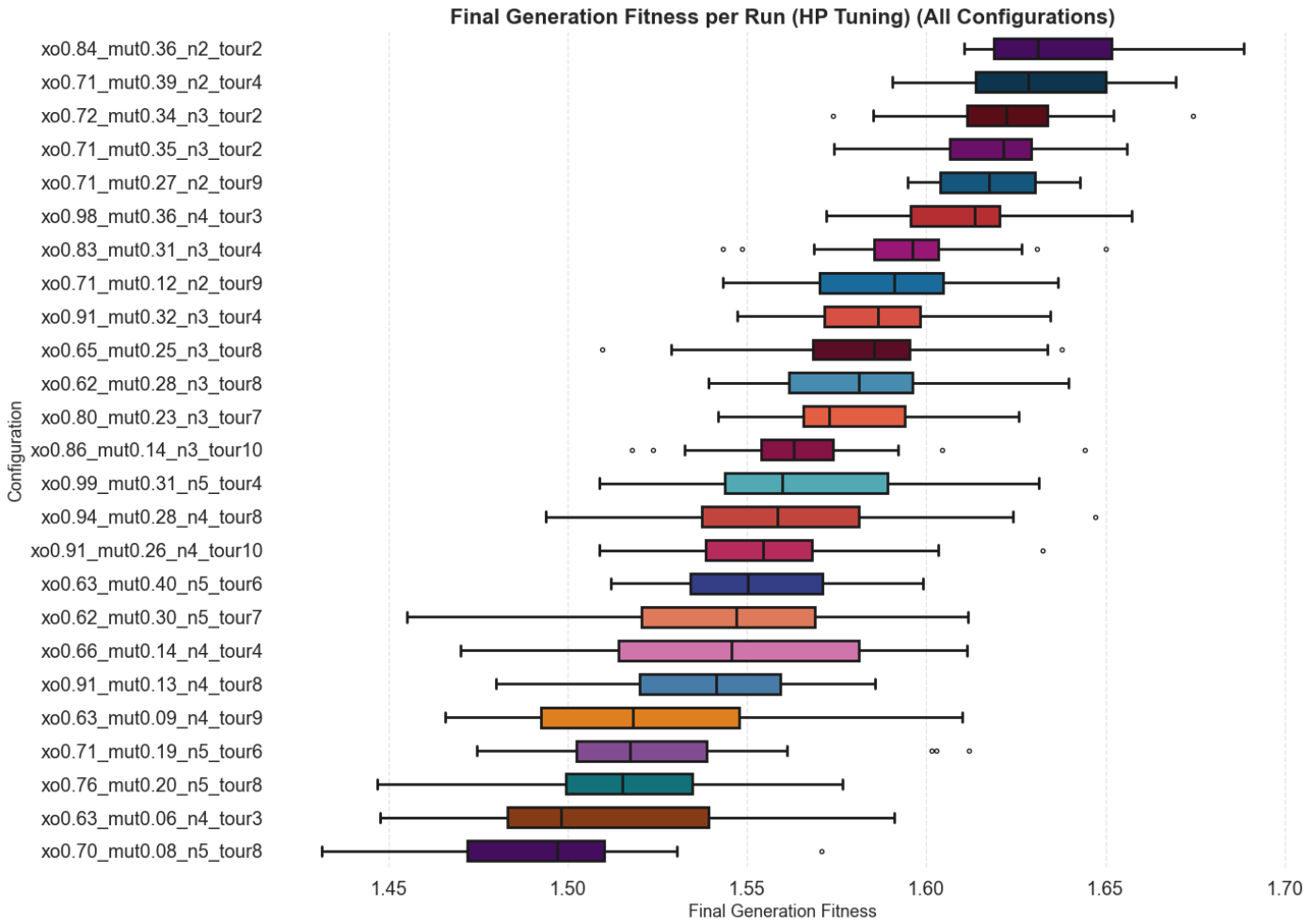


Figure 19 – Fitness Across Generations for All Hyperparameter Search Configurations

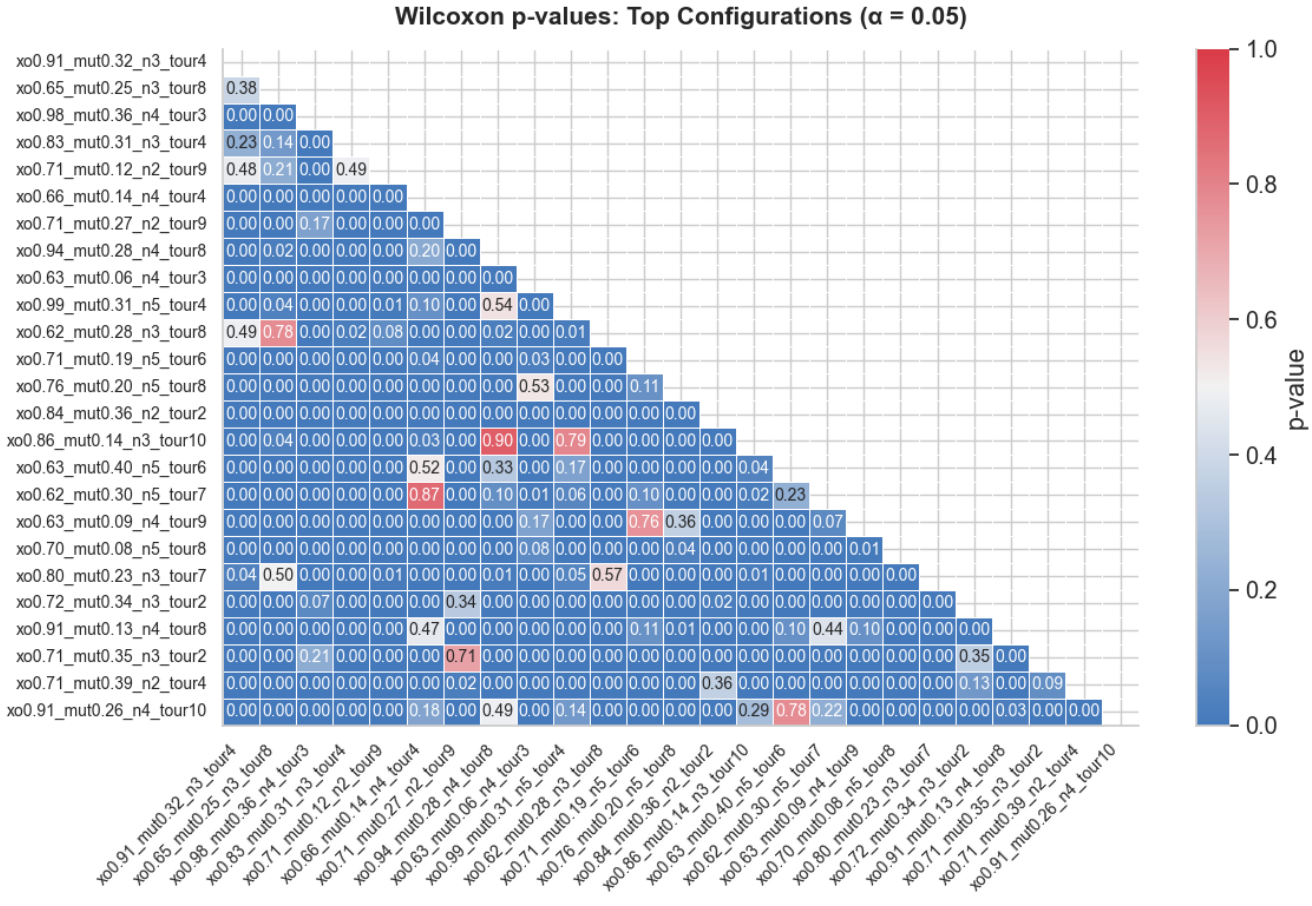


Figure 20 - Wilcoxon Signed-Rank Test p-values for Hyperparameter Tuning Combinations

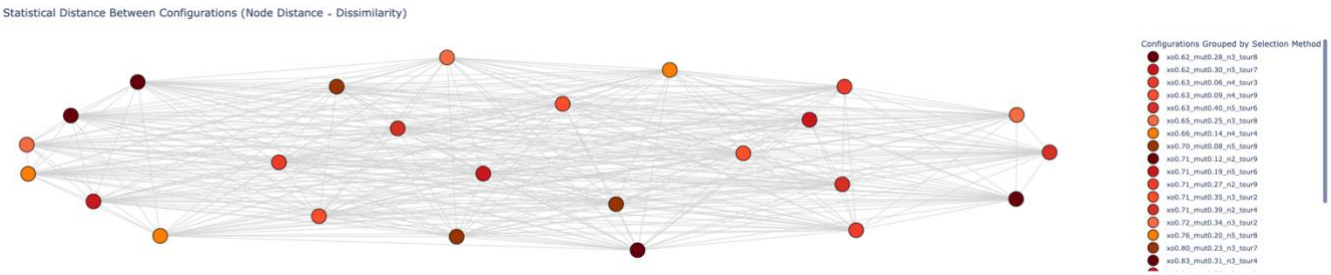


Figure 21 - Network Graph - Statistical Significance Representation

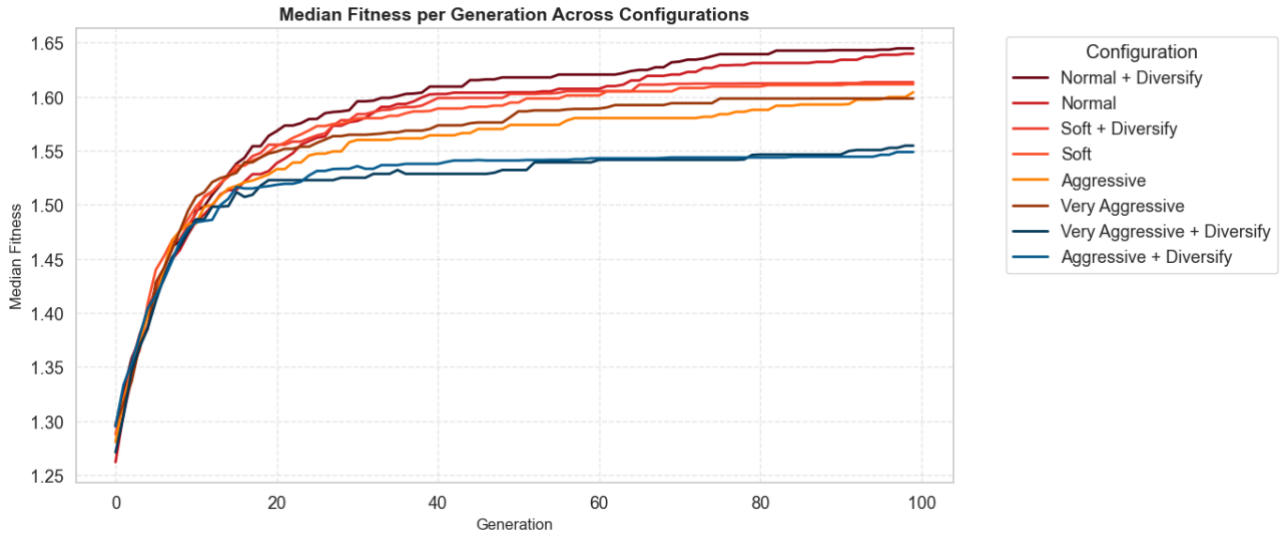


Figure 22 - Fitness Across Generations within Configurations

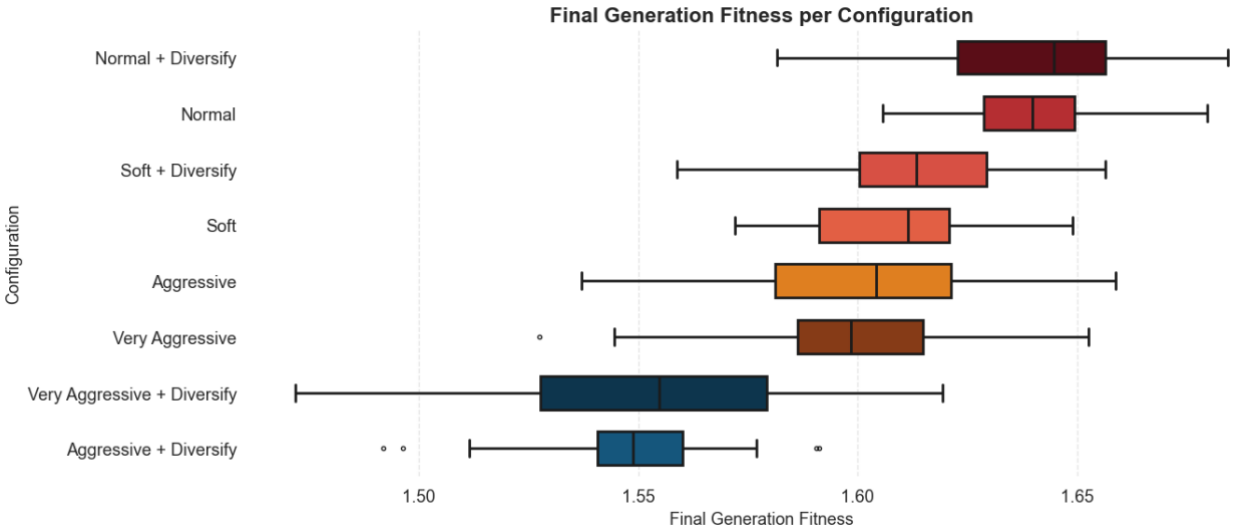
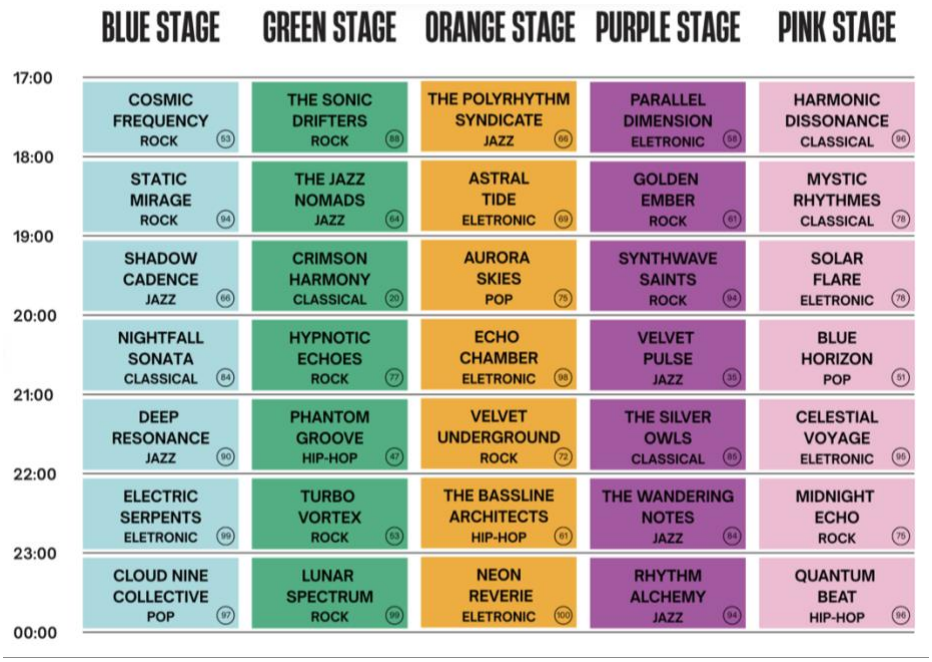


Figure 23 - Variance of Final Generation Fitness within Configurations



	BLUE STAGE	GREEN STAGE	ORANGE STAGE	PURPLE STAGE	PINK STAGE
17:00	COSMIC FREQUENCY ROCK 83	THE SONIC DRIFTERS ROCK 88	THE POLYRHYTHM SYNDICATE JAZZ 66	PARALLEL DIMENSION ELETRONIC 58	HARMONIC DISSONANCE CLASSICAL 96
18:00	STATIC MIRAGE ROCK 94	THE JAZZ NOMADS JAZZ 64	ASTRAL TIDE ELETRONIC 69	GOLDEN EMBER ROCK 61	MYSTIC RHYTHMES CLASSICAL 78
19:00	SHADOW CADENCE JAZZ 66	CRIMSON HARMONY CLASSICAL 20	AURORA SKIES POP 75	SYNTHWAVE SAINTS ROCK 94	SOLAR FLARE ELETRONIC 78
20:00	NIGHTFALL SONATA CLASSICAL 84	HYPNOTIC ECHOES ROCK 77	ECHO CHAMBER ELETRONIC 98	VELVET PULSE JAZZ 35	BLUE HORIZON POP 51
21:00	DEEP RESONANCE JAZZ 90	PHANTOM GROOVE HIP-HOP 47	VELVET UNDERGROUND ROCK 72	THE SILVER OWLS CLASSICAL 85	CELESTIAL VOYAGE ELETRONIC 95
22:00	ELECTRIC SERPENTS ELETRONIC 99	TURBO VORTEX ROCK 53	THE BASSLINE ARCHITECTS HIP-HOP 61	THE WANDERING NOTES JAZZ 84	MIDNIGHT ECHO ROCK 75
23:00	CLOUD NINE COLLECTIVE POP 97	LUNAR SPECTRUM ROCK 99	NEON REVERIE ELETRONIC 100	RHYTHM ALCHEMY JAZZ 94	QUANTUM BEAT HIP-HOP 96
00:00					

Figure 24 - Final Solution

ANNEX B: TABLES

Operator Type	Operator	Default Parameter(s)
Selection	Tournament	Tournament size = 4
	Fitness-Proportionate	-
	Ranking	-
Crossover	Partially Mapped	-
	Fitness-Based Slot	-
Mutation	N Swap	N = 2
	Scramble	Maximum segment length = 4
	Prime Slot Swap	-
	Preserve Best Slots	Keep ratio = 0.5

Table 1 - Baseline Genetic Algorithm Configuration Parameters

Parameter	Selected Value	Reasoning
Population Size	50	Common heuristic suggests that larger populations improve performance up to a point of diminishing returns
Number of generations	100	Ensures sufficient evolutionary progress while keeping computational cost reasonable.
Elitism	True	Preserves top-performing individuals across generations, promoting more stable and consistent convergence
Crossover Probability	0.9	Encourages exploration by promoting genetic diversity.
Mutation Probability	0.1	Allows for occasional variation to avoid premature convergence

Table 2 - Genetic Algorithm Configuration Parameters

Parameter	Selected Value
Maximum number of iterations	100
Initial temperature	1.0
Iterations per temperature level	10
Neighborhood function	0.9
Mutation Probability	0.1

Table 3 – Simulated Annealing Configuration Parameters

Parameter	Search Range	Effect
Crossover Probability	0.6 - 1	High values encourage exploration by frequently recombining individuals
Mutation Probability	0.05 – 0.4	Lower values preserve structure and reduce randomness, focusing on exploitation
Tournament Size	2 - 10	Higher values increase selection pressure, but reduce diversity and increase computation
Number of swaps	2 - 5	Lower values promote small, incremental changes

Table 4 – Parameter Search Space in Genetic Algorithm with Tournament Selection, N Swap mutation and Partially Mapped Crossover

REFERENCES

[Bib. 1] mgalao. "GitHub - Mgalao/computational-intelligence-for-optimization-project" GitHub, 2025.

[Bib. 2] Vanneschi, L., & Silva, S. (2023). Lectures on intelligent systems. Springer.

[Bib. 3] Mitchell, M. (1998). An introduction to genetic algorithms. MIT Press.

[Bib. 4] Bäck, T. (1996). Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press.

[Bib. 5] Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins (Ed.), Foundations of Genetic Algorithms (pp. 69–93). Morgan Kaufmann.

[Bib. 6] Wikipedia contributors. (n.d.). Crossover (genetic algorithm). Wikipedia. Retrieved May 23, 2025, from https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)