Formal Definition of the Problem:

**Clearly define the optimization problem, describe how an individual is represented, define the search space and explain the fitness function.**

The goal of this project is to create a festival schedule that distributes artist. Across different stages and time slots in the most balanced and interesting way possible. Each individual in the genetic algorithm represents a full festival lineup which is a list where each artist is assigned to a stage and a slot.

The search space is huge because there are so many ways to combine artists slots, so doing it exhaustively just wouldn’t be practical. That’s where the genetic algorithm comes in, helping us explore smarter options.

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.

**Detailed Description of Implemented Selection and Genetic Operators:**

In this project, we used different types of selection, mutation, and crossover operators to guide the genetic algorithm toward better festival lineups over generations.

Selection Mechanisms:

We implemented two selection methods to choose which individuals get to reproduce:

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

Crossover Operators:

To create new individuals, we combined genes (artist-slot assignments) from two parents using:

* One-Point Crossover: We split both parents at a random point and swap the second halves. It’s simple but effective for mixing two different solutions.
* Uniform Crossover: For each gene (artist assignment), we randomly pick whether it comes from parent A or parent B. This results in a more even blend and promotes exploration of the search space.

Both crossover methods help generate new lineups that might inherit the best parts of their “parents”, but also introduce variation so the population doesn’t get stuck repeating itself.

Mutation Operators:

To keep the population evolving and avoid premature convergence, we added several mutation strategies:

* Swap Mutation: Randomly picks two artists and swaps their assigned time slots or stages.
* Random Resetting: Randomly reassigns a new stage or time slot to one artist.
* Block Shuffle Mutation: Selects a group (block) of artists and shuffles their assignments among themselves. It is a more disruptive mutation that can create bigger structural changes.

Each mutation operator adds a different kind of randomness. Some are subtle tweaks, others more drastic. Together they help explore more of the solution space and potentially escape local optima.

These operators, when combined properly, allow the algorithm to evolve increasingly better schedules over time, balancing exploration (trying new solutions) and exploitation (refining what works).

● Performance Analysis: Comparison of different implementations and analysis of how they affect the performance.

**● Justification of Decisions**

○ Why did you choose this representation?

We chose this representation because it offers a straightforward and flexible way to model the entire festival lineup. Each individual in the population is represented as a list of tuples, where each tuple contains an artist’s ID along with the assigned stage and time slot. This structure is simple to work with and easy to manipulate during crossover and mutation operations. It also makes it easy to check constraints (like avoiding conflicts or double bookings), and to calculate things like genre distribution or prime time assignments. In other words, it’s a representation that maps directly to the real-world problem while also working smoothly with the genetic algorithm logic. It balances clarity, practicality, and compatibility with evolutionary operators, which made it the best fit for this optimization problem.

○ How did you design the fitness function?

The fitness function was designed to reflect the main goals of the festival scheduling problem: minimize artist conflicts, reward placing popular artists in prime-time slots, and promote genre diversity. We broke the problem down into these three core components and calculated a score for each one individually. For conflicts, we used a conflict matrix that penalizes lineups where artists with overlapping fan bases perform at the same time. For popularity, we gave bonus points when highly popular artists were scheduled during peak hours. And for genre diversity, we encouraged a mix of different musical styles across stages and time blocks. Each of these components was normalized so they could be combined, and we assigned weights to balance their importance. The final fitness score is a weighted sum of these values. This structure gave me flexibility to adjust priorities and allowed the genetic algorithm to evolve solutions that weren’t just technically valid but also aligned with what makes a real-life festival lineup appealing.

To make sure the fitness function reflected the real priorities of the problem, we normalized each component to a common scale (between 0 and 1) so they could be combined fairly. For example, the conflict score was converted into a ratio of total possible conflicts, and the popularity score was normalized based on the maximum achievable popularity in prime-time slots. The same was done for genre diversity, considering the proportion of unique genres per stage or time block. After normalization, we assigned weights to each component based on how important we considered them to the overall festival experience. Since avoiding artist conflicts was critical, that term received the highest weight. Prime time popularity came next, as it strongly affects audience satisfaction and engagement. Genre diversity, while still important, received a slightly lower weight to avoid penalizing lineups that were strong in other areas.

○ Which configurations performed best?

* How many did you test?
* How did you measure success?

○ How do different operators influence GA convergence?

○ Have you implemented elitism? What impact does it have?

○ Did your approach yield good results? What improvements could be made?