**Genetic Algorithms**

**Initial Implementation**

The basic implementation follows the core principles of genetic algorithms to solve the classical 8 Queens puzzle, where eight queens must be placed on a chessboard such that no queen can attack another. My implementation uses a relatively straightforward encoding where each individual in the population represents a potential solution, encoded as an array of 8 integers, where each integer represents the row position of a queen in that column.

The key components of the implementation include:

1. State Representation and Fitness Calculation The EightQueensState class encapsulates the board state and fitness calculation. Each state is represented as a 1D numpy array where the index represents the column, and the value represents the row position of the queen. The fitness function calculates the number of non-attacking pairs of queens, with a goal fitness value of 28 (maximum possible non-attacking pairs for 8 queens).
2. Population Initialization The algorithm implements two initialization strategies:

* Random: Generating completely random queen positions
* Neighbors: Creating a population from random neighbors of an initial state This dual approach helps balance between exploration and exploitation from the start.

1. Selection Mechanism The selection process uses fitness proportionate selection (roulette wheel selection), where:

* Each individual's selection probability is proportional to its fitness
* A cumulative probability approach is used to select parents
* Higher fitness individuals have a better chance of being selected for reproduction

1. Genetic Operators The implementation uses two primary genetic operators: a) Crossover: Single-point crossover where:
   * A random crossover point is selected
   * Two parents exchange genetic material to produce two offspring b) Mutation: Random mutation with a configurable probability where:
   * A queen's position in a randomly selected column may change
   * The new position is chosen from available rows, excluding the current position

The algorithm continues until a solution is found (fitness equals goal\_fitness) or the maximum number of generations is reached. This implementation consistently finds solutions within the 30-second requirement, demonstrating the effectiveness of genetic algorithms for constraint satisfaction problems.