Implementation Plan

Step 1: Define the Problem and Constraints

1. **Problem Definition**: Formulate the exam timetabling problem by specifying the number of courses, students, rooms, timeslots, and other relevant parameters.
2. **Constraints**:
    - **Hard Constraints**:
        - No student should have overlapping exams.
        - Room capacity must not be exceeded.
        - Rooms must have the required facilities.
        - Only one exam per room per timeslot.
    - **Soft Constraints**:
        - Minimize the proximity of exams.
        - Schedule exams during preferred times and rooms.

Step 2: Data Preparation

1. **Input Data**: Prepare and load the input data, such as student-course enrollments, room capacities, and facilities.
2. **Matrix Initialization**: Initialize matrices and arrays representing the problem constraints, such as conflict matrices and capacity arrays.

Step 3: Initialize Population

1. **Generate Initial Solutions**: Create an initial population of solutions (timetables). Each solution should be randomly generated and may be infeasible.
2. **Evaluate Initial Solutions**: Calculate the fitness of each solution based on the violation of constraints.

Step 4: Normalize Constraint Violations

1. **Identify Violations**: For each solution, identify and quantify the constraint violations.
2. **Calculate Maximum Violation**: Determine the maximum possible violation for each type of constraint.
3. **Normalize Violations**: Normalize the violations by dividing the actual violation by the maximum possible violation.
4. **Aggregate Normalized Violations**: Sum the normalized violations to get a single score for each solution.

Step 5: Identify and Sort Infeasible Solutions

1. **Identify Infeasibility**: Mark solutions that violate any constraints as infeasible.
2. **Sort Solutions**: Sort infeasible solutions based on their normalized constraint violation scores.

Step 6: Repair Function

1. **Select Repair Candidates**: Select the infeasible solutions with the lowest normalized constraint violation for repair.
2. **Replace Procedure**:
    - For each infeasible variable, select donor solutions that are closest in Euclidean distance.
    - Replace infeasible variables with feasible variables from the donor solution.
3. **Iterate and Repair**: Repeat the repair process for a set number of iterations or until all solutions are feasible.

Step 7: Genetic Algorithm Operations

1. **Selection**: Select parent solutions for crossover based on their fitness.
2. **Crossover**: Combine pairs of parent solutions to produce offspring solutions.
3. **Mutation**: Introduce random changes to some offspring solutions to maintain genetic diversity.
4. **Repair**: Apply the repair function to any infeasible offspring solutions.
5. **Evaluate Fitness**: Calculate the fitness of the offspring solutions.

Step 8: Integration and Iteration

1. **Population Update**: Replace the old population with the new population of offspring solutions.
2. **Repeat**: Repeat the selection, crossover, mutation, repair, and evaluation steps for a specified number of generations or until a satisfactory solution is found.

Step 9: Final Solution

1. **Select Best Solution**: From the final population, select the solution with the best fitness score as the optimal timetable.
2. **Output Results**: Output the final timetable, including the assignment of courses to timeslots and rooms.