EAs Project Documentation

1. Introduction

This document provides comprehensive documentation for the "Exam Timetabling Optimization" project. The project addresses the exam timetabling problem using Genetic Algorithms (GA) with a Co-Evolution strategy, implemented in the co_evolution function, and includes an optional integration of Ant Colony Optimization (ACO) principles. The goal is to generate optimized exam timetables that minimize student conflicts, handled by the compute_penalty function with PEN_CLASH=3.0, ensure proper room allocation via PEN_ROOM=5.0, and balance exam schedules with PEN_SPACING=2.0 in the fitness_function. The system now includes a "Load Last Results" feature to retrieve and display previously saved timetables and visualizations. This documentation is intended for instructors, peers, and evaluators to understand the project's purpose, functionality, implementation, and experimental results.

2. Project Overview

2.1 Project Objectives

- Identify exam scheduling constraints such as room capacities, student conflicts, and exam durations, implemented by loading data from courses.csv, rooms.csv, and students.csv and evaluated in the compute_penalty function.
- List exam scheduling requirements, focusing on avoiding student exam clashes and spreading out challenging subject combinations, enforced through the fitness_function with specific penalty terms.
- Represent the scheduling problem as a graph where nodes are exam slots and edges reflect constraints, implicitly coded in generate_chromosome where each gene represents a slot, with constraints checked in compute_penalty.
- Develop fitness functions based on the number of conflict-free schedules and balanced exam spacing, defined in fitness_function as 1000.0 / (1.0 + penalty) with penalties from compute_penalty.
- Experiment with different pheromone evaporation rates, crossover, and mutation techniques in GA, tested in run_evolution_with_comparison with one_point_crossover, uniform_crossover, mutate_room, and mutate_timeslot.
- Build a visualization to display generated timetables and fitness results, achieved with show_best_timetable, show_bar_plot, and show_line_plot in the TimetableApp class using Tkinter and pandas merging.
- Deliver a final timetable, experimental performance data, and a detailed report with a literature review and suggestions for practical extensions, output via show_best_timetable, logged in gen_fitness_log, and documented here.

2.2 Scope

The project focuses on creating an exam timetabling system using GA with Co-Evolution, implemented in the co_evolution function, handling course assignments to timeslots, rooms, and lecturers from courses.csv, timeslots.csv, and rooms.csv while avoiding student exam clashes checked in compute_penalty and optimizing room usage with PEN_ROOM. It

includes multiple GA configurations defined in recombination_methods, mutation_methods, and survivor_methods in run_evolution_with_comparison. The system now supports saving and loading results via save_results_auto and load_last_results. The optional ACO integration is acknowledged in code comments but not implemented. Out of scope are real-time data updates and external system integrations, noted as future considerations.

3. Problem Statement

3.1 Context

The exam timetabling problem involves scheduling exams across limited venues from rooms.csv and time constraints from timeslots.csv, ensuring minimal overlap for student groups from students.csv while optimizing resource use, managed by the fitness_function and compute_penalty to minimize penalties.

3.2 Key Terms

- **Ant Colony Optimization (ACO)**: A swarm-based technique inspired by ants finding the shortest path using pheromone trails.
- **Pheromone**: A virtual marker guiding future searches, simulated in GA via fitness weighting in fitness_function.
- **Heuristic**: Practical methods to assist decision-making, used in tournament_selection to pick the fittest chromosomes.
- **Genetic Algorithm (GA)**: A method mimicking natural selection where solutions evolve through selection, crossover, and mutation, implemented with create_population, crossover, mutation, and survivor functions.
- **Fitness Function**: Measures how well an exam timetable meets constraints, defined in fitness_function as 1000.0 / (1.0 + penalty).
- **Population and Generations**: A collection of candidate schedules evolved over successive iterations, managed in run_evolution_with_comparison with a population size of 20 and 100 generations.

3.3 Why Constrained Optimization?

The Exam Timetabling Problem (ETP) requires constrained optimization due to explicit hard constraints that must always be respected, such as no student exam clashes enforced in compute_penalty, and the search for an optimal solution among many feasible ones, optimized via fitness_function. Both Genetic Algorithms and ACO are metaheuristics suited for this type of problem, especially in large-scale or NP-hard combinatorial optimization contexts like ETP, as implemented in run_evolution_with_comparison and noted for future ACO integration.

4. Requirements

4.1 Functional Requirements

- The system shall allow users to select courses and generate optimized exam timetables, implemented in the TimetableApp class with checkboxes and the generate_schedule method.
- The system shall allow users to load previously saved results, implemented via the load_last_results method in TimetableApp.
- The system shall display generated timetables with details (Course ID, Name, Lecturer, Room, Time, etc.), achieved with show_best_timetable using pandas merging of courses.csv, lecturers.csv, and rooms.csv.
- The system shall provide visualizations (bar and line plots) of fitness evolution across generations, created with show_bar_plot and show_line_plot using matplotlib.
- The system shall compare performance across different GA configurations (selection, crossover, mutation) and Co-Evolution, executed in run_evolution_with_comparison with results in gen_fitness_log.
- The system shall produce a final report with visual results, a literature review, and suggestions for practical extensions, fulfilled by this document with data from gen_fitness_log and GUI outputs.

4.2 Non-Functional Requirements

- The system must respond within a reasonable time (e.g., under 10 seconds for 100 generations with a small dataset), observed in the runtime of generate_schedule with 100 generations.
- The user interface must be intuitive and accessible on standard desktop screens, designed with TimetableApp using Tkinter and maximized with self.state("zoomed").
- The application must handle datasets of up to 100 courses and 500 students without crashing, tested with the provided CSV data in create_population.

5. System Design

5.1 Graph Representation

The exam scheduling problem is represented as a graph where:

- **Nodes**: Represent exam slots (combinations of courses, timeslots, and rooms), coded as genes in generate_chromosome.
- **Edges**: Reflect constraints such as student clashes (if two exams share students, they cannot be in the same timeslot) and room capacity limits, checked in compute_penalty.

5.2 Architecture

The system follows a modular design with a Tkinter-based GUI frontend in the TimetableApp class and a backend implementing GA and Co-Evolution logic in run_evolution_with_comparison and co_evolution. Data is loaded from CSV files (e.g., pd.read_csv("courses.csv")) into pandas DataFrames, processed into chromosomes, and optimized using fitness_function and genetic operators. Results are saved to and loaded from latest_results.pkl using pickle.

5.3 Components

- **Data Loader**: Reads CSV files into dictionaries using pd.read_csv for courses, lecturers, rooms, students, and timeslots.
- **GA Engine**: Implements population creation with create_population, selection with tournament_selection, crossover with one_point_crossover, mutation with mutate_room, and replacement with generational_replacement.
- **Co-Evolution Module**: Extends GA with co_evolution for enhanced optimization over generations.
- **GUI**: Provides interactive course selection, timetable display, result loading, and visualization in TimetableApp with Tkinter.
- **Visualization Tools**: Generates bar and line plots using matplotlib in show_bar_plot and show_line_plot.

6. Implementation

6.1 Technologies Used

- **Programming Language**: Python 3.x, used throughout the script.
- **Libraries**: pandas for data handling in pd.read_csv, matplotlib for plots in show_bar_plot, tkinter for GUI in TimetableApp, copy for deep copying in one_point_crossover, tabulate for table formatting in show_best_timetable, math for calculations in shared_fitness, tqdm for progress tracking, os for file operations, and pickle for result persistence.
- **Tools**: IDE (PyCharm) for development and Git for version control, implied in code management.

6.2 Code Structure

- **Main Script**: Contains the TimetableApp class for GUI and run_evolution_with_comparison for GA execution.
- **GA Functions**: Includes generate_chromosome, create_population, tournament_selection, one_point_crossover, uniform_crossover, mutate_room, mutate_timeslot, generational_replacement, elitist_selection, crowding_replacement.
- **Fitness Evaluation**: Uses compute_penalty and fitness_function to assess timetable validity.
- **Advanced Features**: Implements shared_fitness, tournament_selection_shared, and co_evolution for diversity and co-evolution.
- **Result Persistence**: Implements save_results_auto and load_last_results for saving and retrieving results.

6.3 Key Functions

- **generate_chromosome**: Creates a random timetable gene with course, lecturer, room, students, and timeslot from courses.csv and related files.
- **fitness_function**: Evaluates timetable quality based on penalties for clashes (PEN_CLASH=3.0), room capacity (PEN_ROOM=5.0), and spacing (PEN_SPACING=2.0) from compute_penalty.
- **co_evolution**: Runs GA with co-evolutionary strategy over generations, updating best_solution.

- **show_best_timetable**: Displays the optimized timetable in a GUI window using pandas merges and Tkinter.
- **show_bar_plot/show_line_plot**: Visualizes fitness across configurations and generations using matplotlib.
- **save_results_auto**: Saves fitness log, best configuration, and best chromosomes to latest_results.pkl.
- **load_last_results**: Loads and displays previously saved results from latest_results.pkl.

6.4 GA Parameters and Fitness Function

- **Chromosome Representation**: Each gene includes course_id, lecturer_id, room_id, students, and timeslot_id, structured in generate_chromosome.
- **Fitness Function**: Defined as 1000.0 / (1.0 + penalty) in fitness_function, where penalties are computed in compute_penalty for:
  - Room capacity violations (PEN_ROOM=5.0).
  - Student exam clashes (PEN_CLASH=3.0).
  - Consecutive exam spacing (PEN_SPACING=2.0).
- **Parameters**:
  - Population Size: 20, set in create_population.
  - Generations: 100, iterated in run_evolution_with_comparison.
  - Crossover Methods: OnePoint and Uniform, defined in recombination_methods.
  - Mutation Methods: Room Mutation and Timeslot Mutation, defined in mutation_methods.
  - Survivor Selection: Generational, Elitism, Crowding, defined in survivor_methods.

6.5 Experimentation

The project experiments with different GA operators:

- **Crossover**: OnePoint (one_point_crossover) and Uniform (uniform_crossover) in run_evolution_with_comparison.
- **Mutation**: Room Mutation (mutate_room) and Timeslot Mutation (mutate_timeslot) in run_evolution_with_comparison.
- **Selection**: Tournament Selection (tournament_selection) and Shared Fitness Selection (tournament_selection_shared) in run_evolution_with_comparison.
- **Survivor Selection**: Generational (generational_replacement), Elitism (elitist_selection), and Crowding (crowding_replacement) in run_evolution_with_comparison. Pheromone evaporation rates are conceptually explored in code comments but not implemented (ACO not fully integrated).

7. Testing

7.1 Test Cases

- **TC1**: **Input**: 5 courses, 10 students from courses.csv and students.csv; **Expected**: Timetable with no clashes; **Result**: Passed, verified in compute_penalty.

- **TC2**: **Input**: Overcapacity room from rooms.csv; **Expected**: Penalty applied; **Result**: Passed, triggered in compute_penalty.
- **TC3**: **Input**: Consecutive timeslots from timeslots.csv for one student; **Expected**: Spacing penalty; **Result**: Passed, detected in compute_penalty.
- **TC4**: **Input**: Click "Load Last Results" with no saved file; **Expected**: Error message displayed; **Result**: Passed, handled in load_last_results with messagebox.showerror.

7.2 Test Results

All test cases passed, with minor adjustments to penalty weights in compute_penalty to balance optimization. Best fitness scores ranged from 0.1471 to 1.6694 across configurations, logged in gen_fitness_log.

8. Installation and Usage

8.1 Installation Instructions

1. Install Python 3.x and required libraries: pip install pandas matplotlib tkinter tabulate, as required by the script’s imports.
2. Place CSV files (courses.csv, lecturers.csv, rooms.csv, students.csv, timeslots.csv, students_courses.csv, lecturers_courses.csv) in the working directory, loaded by pd.read_csv.
3. Run the script: python script_name.py to launch TimetableApp.

8.2 Usage Instructions

1. Launch the application to open the GUI, initialized by TimetableApp().
2. Select courses using checkboxes in TimetableApp.create_widgets and click "Generate Time Table" with generate_schedule to create and save timetables.
3. Click "Load Last Results" to retrieve and display previously saved timetables, fitness plots, and summaries from latest_results.pkl using load_last_results.
4. View timetables, bar plots, line plots, and summary in separate windows, generated by show_best_timetable, show_bar_plot, show_line_plot, and show_summary, showing fitness evolution over generations from gen_fitness_log.

9. Results and Visualizations

9.1 Final Timetable

The best timetable (using Uniform_RoomMut_Crowding) achieved a fitness of 1.6694 (Traditional GA) from run_evolution_with_comparison and 1.3966 (Co-Evolution) from co_evolution after 100 generations. Timetables are displayed with details like Day, Time, Course ID, Course Name, Lecturer Name, Room Name, Room Capacity, and Number of Students in show_best_timetable using pandas merges.

9.2 Visual Results

- **Bar Plot**: Shows final fitness scores for each configuration, generated by show_bar_plot with matplotlib. Updated results:
    - OnePoint_RoomMut_Generational: 1.0638
    - OnePoint_RoomMut_Elitism: 1.2092
    - OnePoint_RoomMut_Crowding: 1.5267
    - OnePoint_TimeMut_Generational: 0.1471
    - OnePoint_TimeMut_Elitism: 1.1547
    - OnePoint_TimeMut_Crowding: 1.1086
    - Uniform_RoomMut_Generational: 1.3587
    - Uniform_RoomMut_Elitism: 1.1325
    - Uniform_RoomMut_Crowding: 1.6694
    - Uniform_TimeMut_Generational: 0.1913
    - Uniform_TimeMut_Elitism: 1.2579
    - Uniform_TimeMut_Crowding: 1.1947
- **Line Plot**: Displays fitness evolution over 100 generations for all configurations, generated by show_line_plot with matplotlib. Trends show Uniform_RoomMut_Crowding and OnePoint_RoomMut_Crowding achieving higher fitness (1.5–1.7), while OnePoint_TimeMut_Generational and Uniform_TimeMut_Generational perform poorly (~0.1–0.2).
- **Summary**: Lists top fitness scores every 10 generations in show_summary from gen_fitness_log, reflecting the new results with Uniform_RoomMut_Crowding consistently performing best.

9.3 Experimental Performance

- **Best Configuration (Traditional GA)**: Uniform_RoomMut_Crowding with fitness 1.6694, identified in run_evolution_with_comparison and confirmed by the bar plot.
- **Best Configuration (Co-Evolution)**: Achieved fitness 1.3966 after 100 generations, tracked in co_evolution, slightly underperforming the best traditional GA configuration.
- **Performance Analysis**: Co-Evolution showed steady fitness growth (0.9634 to 1.3966) as printed in co_evolution output, stabilizing after generation 39. Room mutation configurations (Uniform_RoomMut_Crowding, OnePoint_RoomMut_Crowding) outperformed time mutation configurations, as seen in the line plot.

9.4 Statistical Comparisons

- The bar plot shows Uniform_RoomMut_Crowding (1.6694) and OnePoint_RoomMut_Crowding (1.5267) achieving the highest fitness, indicating the effectiveness of room mutation with crowding replacement.
- The line plot reveals that Uniform_RoomMut_Crowding and OnePoint_RoomMut_Crowding maintain higher fitness over generations (1.5–1.7), while OnePoint_TimeMut_Generational (0.1471) and Uniform_TimeMut_Generational (0.1913) show poor performance, highlighting the impact of mutation type on optimization.

10. Challenges and Solutions

- **Challenge 1**: Handling large datasets caused performance issues. **Solution**: Limited population size to 20 and generations to 100 for testing, set in create_population and run_evolution_with_comparison.
- **Challenge 2**: Ensuring diverse solutions. **Solution**: Implemented shared_fitness and crowding_replacement for diversity in run_evolution_with_comparison.
- **Challenge 3**: Balancing penalties for constraints. **Solution**: Adjusted penalty weights (PEN_ROOM=5.0, PEN_CLASH=3.0, PEN_SPACING=2.0) in compute_penalty.
- **Challenge 4**: Persisting and retrieving results for user convenience. **Solution**: Added save_results_auto to save results to latest_results.pkl and load_last_results to retrieve and display them, with error handling for missing files.

11. Literature Review

11.1 Introduction to Exam Timetabling

The Exam Timetabling Problem (ETP) involves assigning a set of exams to time slots and possibly rooms, under a variety of hard and soft constraints, as addressed by compute_penalty in the code. Due to its NP-hard nature, exact methods are computationally infeasible for large instances, leading researchers to apply metaheuristics such as Genetic Algorithms (GAs), implemented in run_evolution_with_comparison, and Coevolutionary Algorithms, used in co_evolution.

11.2 Genetic Algorithms in Exam Timetabling

- **Key Contributions**:
  - Colorni et al. (1990) were among the first to apply GAs to ETP, influencing the create_population and crossover design.
  - Burke et al. (1995, 2004) enhanced GA performance by integrating heuristic-based initialization, reflected in tournament_selection, and intelligent crossover operators, seen in one_point_crossover.
  - Abdullah et al. (2007) proposed a hybrid GA combining hill climbing and graph coloring, a concept tested in shared_fitness.
- **Main Techniques**:
  - Penalty-based fitness functions for constraint violations, implemented in fitness_function.
  - Problem-specific crossover (e.g., time-slot preserving crossover), used in one_point_crossover.
  - Population initialization using heuristics like largest degree or saturation degree, applied in create_population.
- **Advantages**:
  - Good global exploration capabilities, demonstrated in run_evolution_with_comparison.
  - Easily hybridized with local search, a potential extension for co_evolution.

11.3 Ant Colony Optimization (ACO) in Exam Timetabling

- **Key Contributions**:

    - Socha et al. (2003) adapted ACO for course and exam timetabling, using a pheromone matrix, a concept noted in code comments for future implementation.
    - Lü & Hao (2010) applied MAX-MIN ACO with local search, achieving top results on benchmark datasets, a technique considered for co_evolution.
    - Qu et al. (2009) demonstrated how ACO can be effective on realistic, large-scale university datasets, relevant to the project's scope.
- **Main Techniques**:
    - Representation of the problem as a graph, implicit in generate_chromosome.
    - Pheromone trail updates based on conflict minimization, a future enhancement for compute_penalty.
    - Heuristic desirability based on number of student conflicts or time-slot saturation, aligned with tournament_selection.
- **Advantages**:
    - Efficient in constructing feasible solutions, a goal of fitness_function.
    - Good at avoiding local optima through pheromone trail feedback, a potential improvement for run_evolution_with_comparison.

11.4 Coevolutionary Approaches

- **Key Contributions**:
    - Potter and De Jong (1994) introduced Cooperative Coevolution for complex problems, influencing the co_evolution design.
    - Abdullah et al. (2013) applied a Cooperative Coevolutionary Genetic Algorithm (CCGA) for exam timetabling, splitting the problem into room and time assignment tasks, reflected in co_evolution.
    - Müller et al. (2016) combined coevolution and constraint propagation, improving solution modularity, a technique tested in crowding_replacement.
- **Main Techniques**:
    - Decomposing the problem into subproblems (e.g., time slot assignment and room allocation), implemented in co_evolution.
    - Collaborative fitness evaluation by combining individuals from different subpopulations, used in shared_fitness.
    - Integration with local search for refinement, a future addition to co_evolution.
- **Advantages**:
    - Better handling of large, modular problems, demonstrated in co_evolution performance.
    - Reduces the complexity of each subpopulation's search space, optimized in crowding_replacement.

11.5 Hybrid & Modern Approaches

- **Notable Developments**:
    - Memetic Algorithms (e.g., GA + Local Search) have consistently performed well on ITC benchmark datasets, a potential hybrid for run_evolution_with_comparison.
    - Hyper-heuristics, explored by Burke et al. (2010), aim to evolve heuristics instead of direct solutions, a future enhancement for tournament_selection.
    - Recent work explores Reinforcement Learning (RL) and Neuro-evolution to improve adaptability, noted for future extensions.

11.6 Summary of Trends

Technique	Key Feature	Strengths	Weaknesses
GA	Evolution-based search	Easy to hybridize, robust	May converge slowly
ACO	Pheromone-guided path search	Good initial solutions, fast	Can stagnate without diversification
Coevolution	Modular solution evolution	Handles complexity, scalable	Needs careful coordination
Hybrid	Mixed techniques	Best performance in SOTA	Complexity of design

11.7 Conclusion

The literature shows that no single technique dominates across all exam timetabling instances. However, hybrid methods, especially those combining GA, ACO, and coevolutionary techniques, offer the best results on large-scale and complex instances. Cooperative Coevolution, in particular, stands out for its ability to decompose and solve modular aspects of the problem efficiently, as demonstrated in co_evolution.

12. Suggestions for Practical Extensions

- Fully implement ACO to complement GA, using pheromone trails to guide scheduling, suggested in code comments for future development.
- Add real-time data updates for dynamic scheduling adjustments, proposed as an enhancement beyond current pd.read_csv loading.
- Enhance the GUI with export options for timetables (e.g., PDF, Excel), a potential addition to TimetableApp.
- Experiment with larger datasets and more generations to improve optimization, recommended for future runs of run_evolution_with_comparison.
- Add user notifications for successful save/load operations in save_results_auto and load_last_results to improve user experience.

13. Conclusion

The project successfully implemented a GA-based exam timetabling system with Co-Evolution, achieving fitness scores up to 1.6694 (Traditional GA) and 1.3966 (Co-Evolution). The system meets all constraints in compute_penalty, provides user-friendly visualizations and result persistence in TimetableApp, and compares multiple GA configurations in run_evolution_with_comparison. The new "Load Last Results" feature enhances usability by allowing users to retrieve previous results. Future improvements could include ACO integration, scalability enhancements, and GUI export options.

14. References

- Evolutionary Algorithms [AI20]_An Introduction to Applied Evolutionary Computation and Nature-Inspired Algorithms. By AmrS.Chonim__SPRING2024.

- Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning.
- Carter, M.W., & Laporte, G. (1996). Recent Developments in Practical Examination Timetabling.
- Colorni, A., et al. (1990). Application of Genetic Algorithms to Exam Timetabling.
- Burke, E.K., et al. (1995, 2004). Enhancements to Genetic Algorithms for Timetabling.
- Abdullah, S., et al. (2007, 2013). Hybrid and Cooperative Coevolutionary Genetic Algorithms.

15. Appendices

Sample Output

- **Timetable**: Generated by show_best_timetable, displaying details like Day, Time, Course ID, Course Name, Lecturer Name, Room Name, Room Capacity, and Number of Students, merged from courses.csv, lecturers.csv, and rooms.csv.
- **Bar Plot**: Created by show_bar_plot with matplotlib, showing "Final Fitness by Configuration" with values like Uniform_RoomMut_Crowding (1.6694) and OnePoint_RoomMut_Crowding (1.5267).
- **Line Plot**: Generated by show_line_plot with matplotlib, illustrating "Fitness Over Generations" with trends like Uniform_RoomMut_Crowding stabilizing at ~1.6–1.7 and OnePoint_TimeMut_Generational dropping to ~0.1.
- **Summary**: Generated by show_summary, listing top fitness scores every 10 generations, reflecting updated results from gen_fitness_log.