AI420: Evolutionary Algorithms

Project Documentation

1. Project Idea in Detail

The goal of this project is to develop a robust and flexible exam timetabling system using Genetic Algorithms (GAs). Exam timetabling is a well-known NP-hard problem, requiring the scheduling of a series of exams (courses) into available time slots and rooms, while satisfying numerous hard and soft constraints. The system must account for room capacity, student enrolment, lecturer availability, exam conflicts, and spread of exams across the week.

This project tackles the problem using a nature-inspired evolutionary approach that evolves a population of candidate timetables over multiple generations. Each individual represents a full exam timetable, and selection, crossover, and mutation operations are used to iteratively improve the solution quality. The fitness function penalises constraint violations and rewards conflict-free and well-distributed schedules.

2. Main Functionalities of the Project

- Importing structured data from CSVs: courses, students, rooms, timeslots, lecturers.
- Generating initial timetables either randomly or using a greedy heuristic.
- Applying Genetic Algorithm operators:
    - Fitness-based parent selection
    - Order-based and cycle-based crossover
    - Scramble and insert mutation
    - Survivor selection using (mu, lambda) and round-robin approaches
- Fitness evaluation using constraint-aware scoring
- Experimentation with different GA configurations
- Visual comparison of results via bar charts and interactive GUI
- Exporting and inspecting best-found timetables

3. Similar Applications in the Market

- **UniTime**: A comprehensive open-source timetabling platform used by universities to manage course, exam, and room schedules.
- **Celcat Timetabler**: A commercial scheduling tool for academic institutions that provides automatic scheduling, real-time updates, and integration with student data systems.
- **AsureSpace**: A broader space management tool that includes exam and room scheduling, using rule-based engines.
- **ExamScheduler**: A cloud-based tool focused on small-to-medium educational institutions, offering constraint-based scheduling and auto conflict resolution.

4. Literature Review

1. **Burke, E. K., & Newall, J. P. (1999).** A multistage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*.
    - Demonstrates the efficacy of hybrid evolutionary approaches for timetabling, with separate stages for feasibility and optimisation.
2. **Paechter, B., Cumming, A., Luchian, H., & Petruzzelli, A. (1998).** Two solutions to the exam timetabling problem using evolutionary methods. *Proceedings of the Genetic and Evolutionary Computation Conference*.
    - Investigates permutation-based representation and compares GA with hill-climbing hybrids.
3. **Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., & Lee, S. Y. (2009).** A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*.
    - Comprehensive overview of heuristic and metaheuristic methods applied to timetabling, including case studies.
4. **Moukrim, A., & Sourd, F. (2008).** A multi-objective simulated annealing approach to exam timetabling. *Computers & Operations Research*.
    - Presents an alternative to GAs, focusing on balancing hard constraints with student fairness.
5. **Colorni, A., Dorigo, M., & Maniezzo, V. (1992).** Genetic algorithms and highly constrained problems: The time-table case. *Parallel Problem Solving from Nature*.
    - Pioneering work that lays the foundation for using GAs for scheduling.

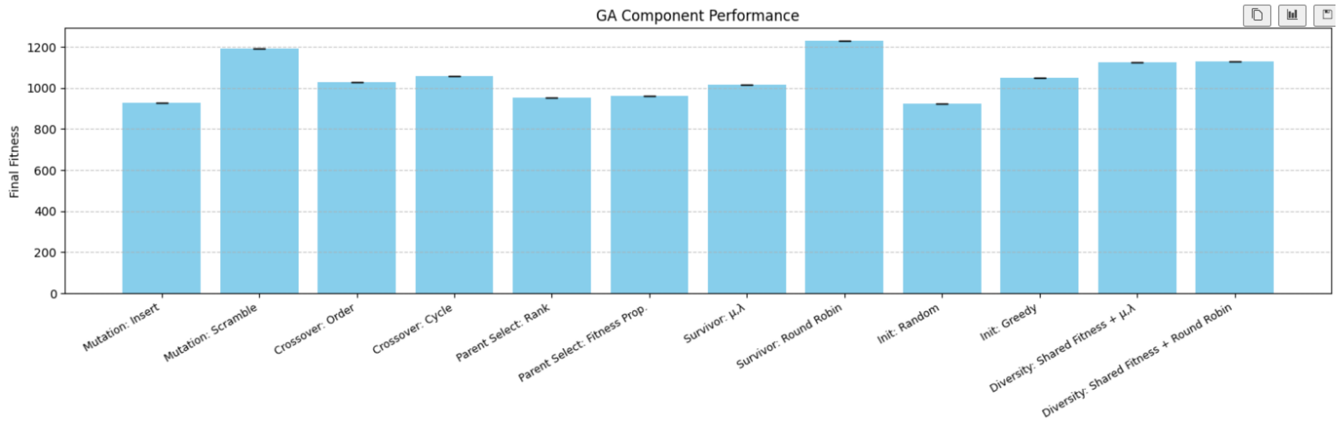
6. **Erben, W. (2001).** A grouping genetic algorithm for graph colouring and exam timetabling. *Practice and Theory of Automated Timetabling*.
    - Focuses on graph-based problem representations and constraint groupings for improved evolution.

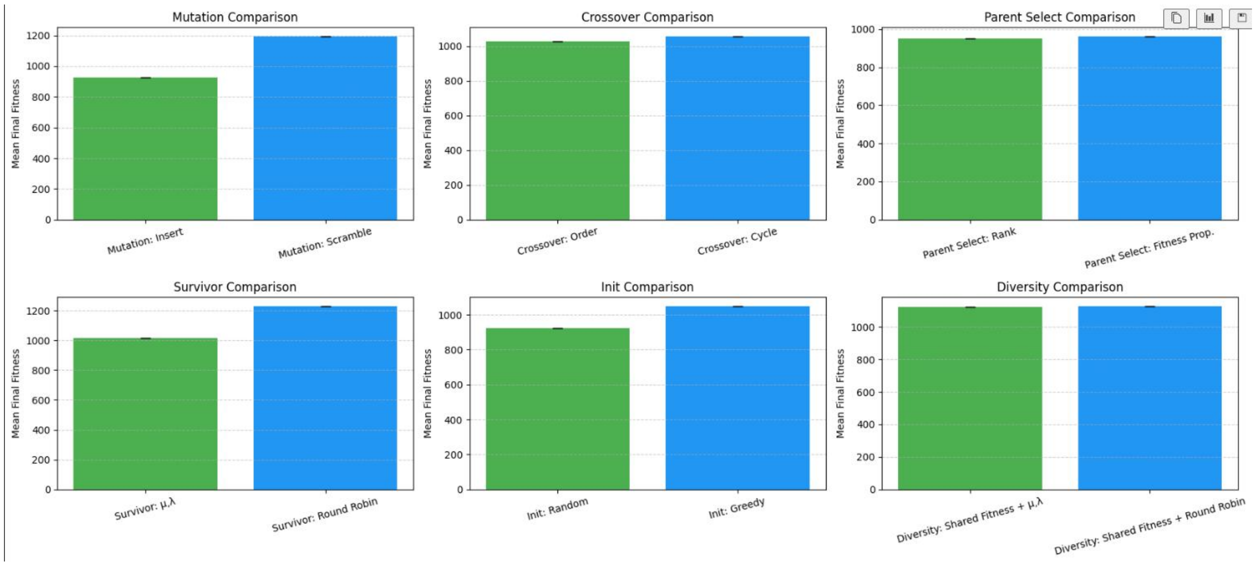
5. Details of Algorithms & Approaches Used

- **Representation**: Each timetable is a list of sessions, with each session defined as a tuple (course_id, lecturer_id, day, timeslot, room_id).
- **Initialisation**:
    - *Random*: Assigns each course a random valid slot and room.
    - *Greedy*: Prefers allocating rooms and slots based on course enrolment and availability.
- **Selection**:
    - *Rank-Based*: Sorts individuals by fitness and selects top ranks with uniform probability.
    - *Fitness-Proportionate*: Selects individuals probabilistically based on scaled fitness values.
- **Crossover**:
    - *Order Crossover*: Preserves relative ordering of sessions to maintain structural feasibility.
    - *Cycle Crossover*: Alternates cycles of genes from both parents to generate diversity.
- **Mutation**: (mutation rate of 0.3 used)
    - *Insert Mutation*: Moves one session to a different position in the sequence.
    - *Scramble Mutation*: Randomly shuffles a sub-segment of the timetable.
- **Survivor Selection**:
    - *(Mu, Lambda)*: Selects the top mu offspring based on fitness.
    - *Round Robin*: Individuals compete in mini-tournaments for survival based on win counts.

- **Fitness Function**: Scores are calculated based on penalties for room overflow, room/time and lecturer/time conflicts, and students having multiple exams in a day. Rewards are given for valid sessions and fairness in distribution.
- **Evaluation Framework**: Multiple configurations are evaluated and compared using average fitness, best and worst scores, and standard deviation. Visualisation is provided via GUI and matplotlib bar charts.

This project demonstrates the ability of Genetic Algorithms to handle complex combinatorial optimisation challenges, and provides a foundation for adaptive, scalable scheduling systems in real-world academic settings.


GA Component Performance
Final Fitness
Mutation: Insert
Mutation: Scramble
Crossover: Order
Crossover: Cycle
Parent Select: Rank
Parent Select: Fitness Prop.
Survivor: μ,λ
Survivor: Round Robin
Init: Random
Init: Greedy
Diversity: Shared Fitness + μ,λ
Diversity: Shared Fitness + Round Robin


Mutation Comparison
Mean Final Fitness
Mutation: Insert
Mutation: Scramble
Crossover Comparison
Crossover: Order
Crossover: Cycle
Parent Select Comparison
Parent Select: Rank
Parent Select: Fitness Prop.
Survivor Comparison
Survivor: μ,λ
Survivor: Round Robin
Init Comparison
Init: Random
Init: Greedy
Diversity Comparison
Diversity: Shared Fitness + μ,λ
Diversity: Shared Fitness + Round Robin