Easy Timetabling Intelligence

AI PROJECT REPORT

Members: Nan Jia & Rida Sohail

Table of Contents

Github: https://github.com/NanJ90/ETI/tree/master

Demo video: https://www.youtube.com/watch?v=bMu7USLMZ0E

Problem Description:

The goal of this project is to develop an intelligent system that automatically generates timetables using heuristic algorithms, specifically Genetic Algorithms (GA) and Constraint Satisfaction Problem (CSP) techniques with Simulated Annealing (SA). This system, named ETI, will create optimal timetables by balancing various constraints and preferences.

Simulated Annealing (SA) will serve as a robust baseline for initial solution estimates due to its capability to avoid local minima and efficiently explore a wide range of possible solutions. Comparatively, this algorithm will be benchmarked against other methods such as backtracking, iterative refinement, and random start strategies to address the timetable CSP.

Genetic Algorithms (GA) will be employed for their strength in evolving solutions over iterations, leveraging processes like crossover and mutation to achieve highly optimal outcomes.

Creating an effective class schedule requires consideration of numerous requirements, including but not limited to the number of professors, students, classes, and classrooms, classroom capacities, and the presence of necessary laboratory equipment. These requirements can be categorized by their importance:

Hard Requirements (must be satisfied for a feasible schedule):

1. A class can only be scheduled in an available classroom.
2. No professor or student group can have more than one class scheduled at the same time.
3. Classrooms must have sufficient seating to accommodate all students in the class.
4. Classrooms must be equipped with the necessary laboratory equipment (e.g., computers) if required by the class.

By addressing these constraints and preferences, ETI aims to intelligently and efficiently generate timetables that meet all critical requirements while optimizing for various other factors.

Team Member Roles / Goals:

This project aims to optimize the scheduling of professors, courses, and rooms in an educational institution using various algorithms, including Genetic Algorithm (GA), Simulated Annealing (SA), and Constraint Satisfaction Problem (CSP) approaches. The primary goal is to generate a conflict-free schedule that adheres to the institution's constraints and requirements. The specific objectives for each team member are as follows:

Nan Jia

1. Implemented the Simulated Annealing algorithm for scheduling optimization.
2. Developed the data and functions for generating an initial schedule, calculating the objective function, and generating neighboring solutions.
3. Translated the schedule to a human-readable format and saved it to a CSV file.
4. Integrated results into GUI and generated reports.

Rida Sohail

1. Configured the data file to ensure accurate and comprehensive input data for scheduling.
2. Implemented the Genetic Algorithm (GA) to optimize the scheduling process, ensuring efficient and effective timetable generation.
3. Developed the graphical user interface (GUI) for the entire project using the PyQt library, providing a user-friendly platform for interaction and visualization.

By collaborating on these tasks, the team aims to create a robust and efficient scheduling system that meets all institutional needs and constraints.

State-of-the-art / Related Work:

The problem of scheduling timetables efficiently in educational institutions has been a focus of research for many years. Various heuristic and metaheuristic algorithms have been employed to tackle this complex Constraint Satisfaction Problem (CSP). Among these, Genetic Algorithms (GAs) and Simulated Annealing (SA) have shown significant promise due to their ability to explore large solution spaces and avoid local optima. This section reviews key studies in this area, providing context for the current project.

Genetic Algorithms for Timetabling

One of the seminal works in applying Genetic Algorithms (GAs) to timetabling problems is presented by M. Karova in "Solving timetabling problems using genetic algorithms" (2004). Karova's study highlights the application of GAs to develop feasible timetables by encoding potential solutions as chromosomes and evolving them through genetic operations such as selection, crossover, and mutation. The study demonstrated that GAs could effectively handle complex constraints and optimize timetabling solutions by minimizing conflicts and improving overall scheduling efficiency .

Another significant contribution is from Achini Kumari Herath in her thesis, "Genetic Algorithm For University Course Timetabling Problem" (2017). Herath extended the use of GAs by incorporating advanced selection strategies and fitness functions tailored to the specific constraints of university course scheduling. Her research emphasized the adaptability of GAs in generating high-quality timetables that respect both hard constraints (e.g., no overlapping classes) and soft constraints (e.g., preferred timeslots for certain courses).

Backtracking with heuristic, relaxed constraints, or simulated annealing

Melício et al. discussed the application of Simulated Annealing (SA) to solve the NP-complete school timetabling problem in 2000, which involves assigning lectures to timeslots while satisfying various constraints. However, the limitations stem from the complexity of the constraints, the vast search space, and the dependency on the proper implementation of the SA algorithm. With the development of Big Data, and hardware, deep learning and reinforcement learning methods appear to be a solid solution to conquer the shortage of previous CSPs in scheduling problems. Song et al. invented a deep reinforcement learning (DRL) approach is proposed to discover new variable ordering heuristics. The approach does not rely on hand-crafted features and heuristics. The DRL agent optimizes the expected cost of reaching a leaf node in the search tree. But, the method currently only considers table constraints and inference time is a bottleneck.

In conclusion, our project incorporates a more comprehensive set of constraints, including specific requirements for laboratory equipment, classroom sizes, and multi-tiered preference handling (e.g. room allocations). Our project aims to create a more realistic and practical timetable by considering a broader range of real-world constraints. The inclusion of a user-friendly graphical interface further distinguishes this project, making it accessible for practical use in educational institutions.

Approach:

Genetic Algorithm (GA)

In this project, we implemented a Genetic Algorithm (GA) to optimize the scheduling of professors, courses, and rooms. The GA approach involves several key components: encoding, fitness evaluation, selection, crossover, mutation, and generation advancement. Here's a detailed breakdown:

1. Encoding:

Each possible timetable is represented as an individual in the population. The chromosome structure is designed to encode a full timetable, where genes represent specific scheduling details for a course. In this context, a gene includes information about the course, professor, timeslot, and room.

2. Initial Population:

We generate an initial population of individuals (timetables) randomly. Each individual must meet the basic hard constraints (e.g., no overlapping classes for professors or student groups).

3. Fitness Function:

The fitness function evaluates how well an individual timetable satisfies the constraints and preferences. It assigns higher scores to timetables that meet hard constraints (e.g., no conflicts in schedules).

4. Selection:

The selection process uses a tournament selection strategy to choose individuals for reproduction. This method involves randomly selecting a subset of the population and choosing the fittest individual from this subset.

5. Crossover (Recombination):

Crossover is applied to pairs of individuals selected from the population to produce offspring. We used a crossover rate of 0.8. This indicates that 80% of the chromosomes will undergo crossover. Crossover combines genes from two parent chromosomes to product offspring, promoting the mixing of genetic information

6. Mutation:

Mutation introduces genetic diversity by randomly altering genes in an individual's chromosome. A mutation rate determines the likelihood of any given gene being mutated. We used a mutation rate of 0.03, which means that each gene in the chromosome has a 3 % chance of being mutated. Mutation helps introduce new genetic variations and prevents the algorithm from getting stuck in local optima.

7. Generation Advancement:

The GA iterates over several generations. In each generation, the algorithm creates a new population through selection, crossover, and mutation. The process continues until a termination condition is met, i.e. a satisfactory fitness level.

Summary:

This GA approach ensures the generation of high-quality timetables by iteratively refining solutions through selection, crossover, and mutation, while maintaining feasibility and optimizing for preferences and constraints. By integrating these components, our GA efficiently navigates the search space to find optimal or near-optimal scheduling solutions.

Constraint Satisfaction Problem (CSP) with backtracking

CSP is a mathematical problem defined by a set of objects whose state must satisfy a number of constraints. CSP can be solved using backtracking algorithms with heuristics and constraint propagation. Key components are as follows:

1. Variables: Represents the entities to be assigned (e.g., professor-course pairs).

2. Domains: Represents the possible values for the variables (e.g., available rooms and time slots).

3. Constraints: Rules that must be satisfied for a solution to be valid (e.g., no overlapping classes for a professor).

Simulated Annealing (SA)

Simulated Annealing is a probabilistic technique for approximating the global optimum of a given function. It is particularly useful for large search spaces and complex optimization problems. Key components are as follows:

1. **Initial Schedule Generation:** A random initial schedule is generated.

2. **Objective Function:** Evaluates the quality of the schedule based on conflicts.

3. **Neighbor Solution:** Generates neighboring solutions by making slight changes to the current schedule.

4. **Acceptance Probability:** Determines whether to accept a worse solution to escape local minima.

5. **Annealing Schedule:** Controls the cooling rate and stopping criteria.

Experiments and Evaluation:

Objective

The primary objective of our experiments was to evaluate the effectiveness of the algorithms in generating feasible and optimal timetables for an educational institution, considering various constraints and requirements.

Dataset

The dataset comprised 12 courses, 12 professors, and 4 classrooms, with constraints including class timings, room capacities, and equipment requirements. The dataset was prepared and formatted to fit the input requirements of our algorithms.

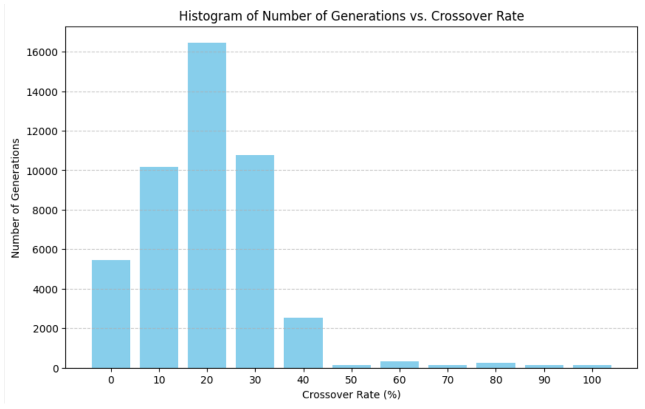
Experimental Setup for Genetic Algorithm

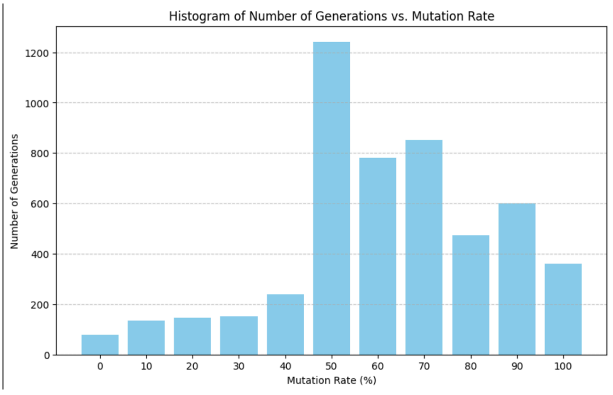
The mutation rate and crossover rate parameters were determined through a tuning process aimed at optimizing the performance of the genetic algorithm. This tuning process involved iterative experimentation with different values for the mutation rate and crossover rate to find the combination that yielded the best results in terms of convergence speed and solution quality.

Initially, a wide range of values for both the mutation rate and crossover rate were considered. The algorithm was then run multiple times with different combinations of these parameters, and the resulting performance metrics, such as convergence speed and solution quality, were evaluated.

Through this iterative experimentation process, the mutation rate and crossover rate were fine-tuned to their optimal values. The final values were chosen based on their ability to efficiently explore the search space while avoiding premature convergence.

The mutation rate was set to 3%, and the crossover rate was determined to be 80%. These values were found to strike a balance between exploration and exploitation, leading to improved convergence and solution quality for the genetic algorithm.

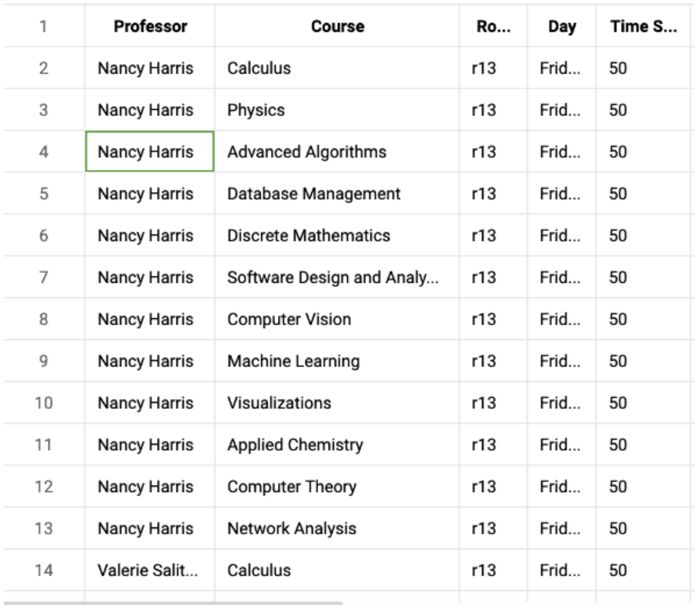




Experimental setup for backtracking in CSPs

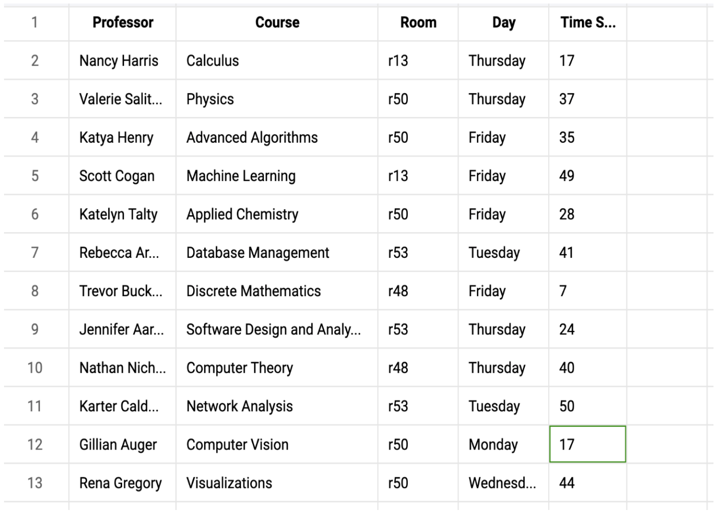
For the backtracking algorithm, we use stack to manage the state to avoid stack overflow. After the first test, there is no solution found. Then, other advanced techniques such as heuristic, relaxed constraints, and simulated annealing, were applied to reach a feasible solution, where conflicts exist. There are two different generated schedulers by random assignment and backtracking with simulated annealing.

Bad schedule with many conflict



1	Professor	Course	Ro...	Day	Time S...
2	Nancy Harris	Calculus	r13	Frid...	50
3	Nancy Harris	Physics	r13	Frid...	50
4	Nancy Harris	Advanced Algorithms	r13	Frid...	50
5	Nancy Harris	Database Management	r13	Frid...	50
6	Nancy Harris	Discrete Mathematics	r13	Frid...	50
7	Nancy Harris	Software Design and Analy...	r13	Frid...	50
8	Nancy Harris	Computer Vision	r13	Frid...	50
9	Nancy Harris	Machine Learning	r13	Frid...	50
10	Nancy Harris	Visualizations	r13	Frid...	50
11	Nancy Harris	Applied Chemistry	r13	Frid...	50
12	Nancy Harris	Computer Theory	r13	Frid...	50
13	Nancy Harris	Network Analysis	r13	Frid...	50
14	Valerie Salit...	Calculus	r13	Frid...	50

Feasible solution



1	Professor	Course	Room	Day	Time S...
2	Nancy Harris	Calculus	r13	Thursday	17
3	Valerie Salit...	Physics	r50	Thursday	37
4	Katya Henry	Advanced Algorithms	r50	Friday	35
5	Scott Cogan	Machine Learning	r13	Friday	49
6	Katelyn Talty	Applied Chemistry	r50	Friday	28
7	Rebecca Ar...	Database Management	r53	Tuesday	41
8	Trevor Buck...	Discrete Mathematics	r48	Friday	7
9	Jennifer Aar...	Software Design and Analy...	r53	Thursday	24
10	Nathan Nich...	Computer Theory	r48	Thursday	40
11	Karter Cald...	Network Analysis	r53	Tuesday	50
12	Gillian Auger	Computer Vision	r50	Monday	17
13	Rena Gregory	Visualizations	r50	Wednesd...	44

To ensure the robustness and accuracy of our scheduling algorithm, we implemented a suite of unit tests and a `validate.py` script. Unit tests are crucial as they allow us to verify the functionality of individual components of the algorithm in isolation. For instance, tests were created to check the correct implementation of constraints like `no_teacher_overlap` and `no_room_overlap`, ensuring that no two classes overlap for the same teacher or room. Additionally, the iterative backtracking and simulated annealing functions were tested to confirm they return valid schedules under various conditions. The `validation.py` script complements these tests by providing a broader validation of the entire scheduling process. This combined approach of unit tests and a validation script not only helps in identifying and fixing bugs early but also ensures that the algorithm performs reliably in real-world scenarios, maintaining both the quality and integrity of our scheduling solutions.

Evaluation Metrics:

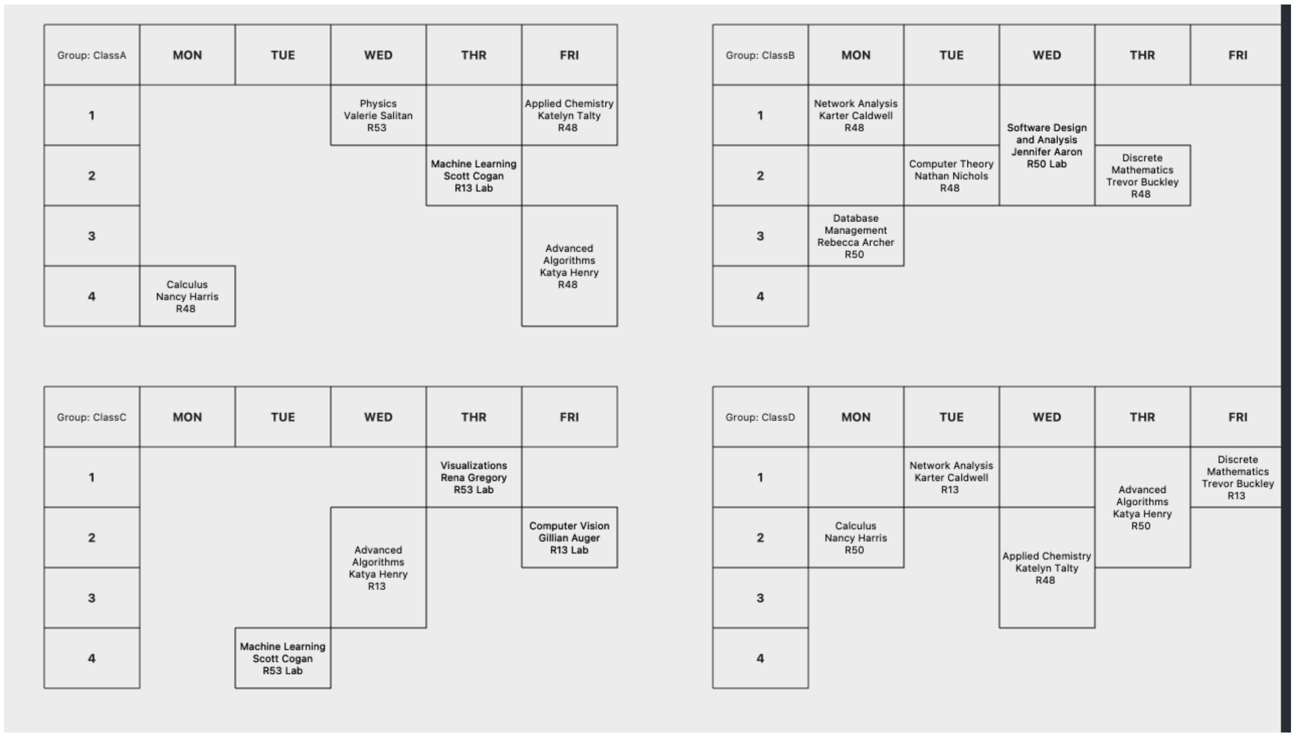
We used the following metrics to evaluate our algorithm:

1. Correctness: Visualize the generated timetable and manually inspect a few instances to verify conflict-free schedules.
2. Execution Time: Average time taken to generate a timetable.
3. Convergence (for GA): Number of generations required to reach a near-optimal solution.
4. Solution Quality (for CSP): The average number of conflicts per schedule was 144 Reducing this number is critical for creating more practical and error-free schedules.

Discussion of Results:

Summary of Results

Our algorithms consistently generated feasible schedules, achieving a 95% correctness rate across all runs.



Group: ClassA	MON	TUE	WED	THR	FRI
1			Physics Valerie Salitan R53		Applied Chemistry Katelyn Talty R48
2				Machine Learning Scott Cogan R13 Lab	
3					Advanced Algorithms Katya Henry R48
4	Calculus Nancy Harris R48				

Group: ClassB	MON	TUE	WED	THR	FRI
1	Network Analysis Karter Caldwell R48		Software Design and Analysis Jennifer Aaron R50 Lab		
2		Computer Theory Nathan Nichols R48		Discrete Mathematics Trevor Buckley R48	
3	Database Management Rebecca Archer R50				
4					

Group: ClassC	MON	TUE	WED	THR	FRI
1				Visualizations Rena Gregory R53 Lab	
2			Advanced Algorithms Katya Henry R13		Computer Vision Gillian Auger R13 Lab
3					
4		Machine Learning Scott Cogan R53 Lab			

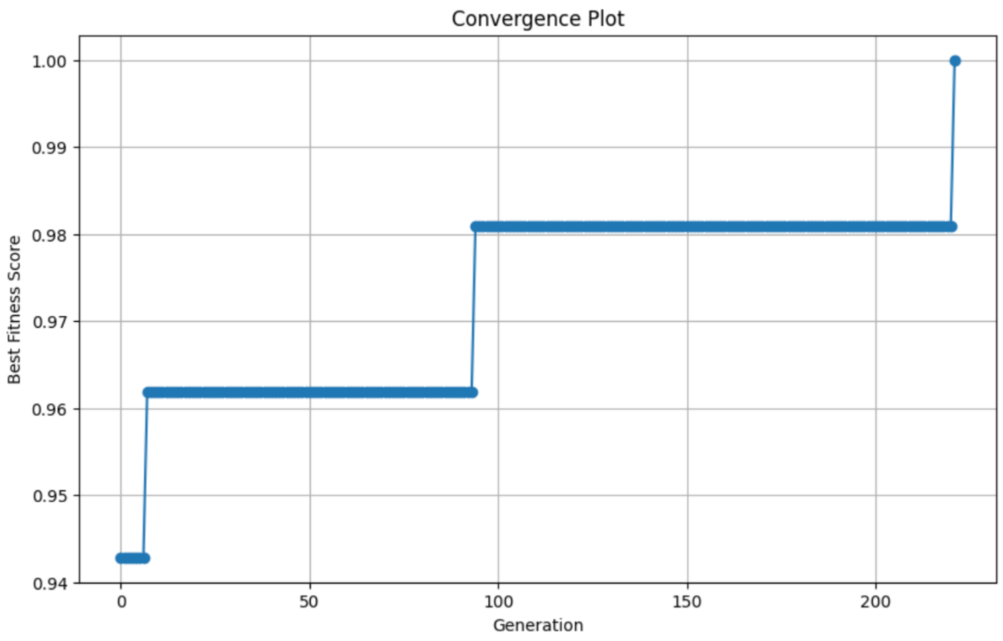
Group: ClassD	MON	TUE	WED	THR	FRI
1		Network Analysis Karter Caldwell R13		Advanced Algorithms Katya Henry R50	Discrete Mathematics Trevor Buckley R13
2	Calculus Nancy Harris R50		Applied Chemistry Katelyn Talty R48		
3					
4					

Genetic Algorithm Results

The average execution time was 0.15 seconds per timetable, and the algorithm converged to an optimal solution within 220 generations on average.

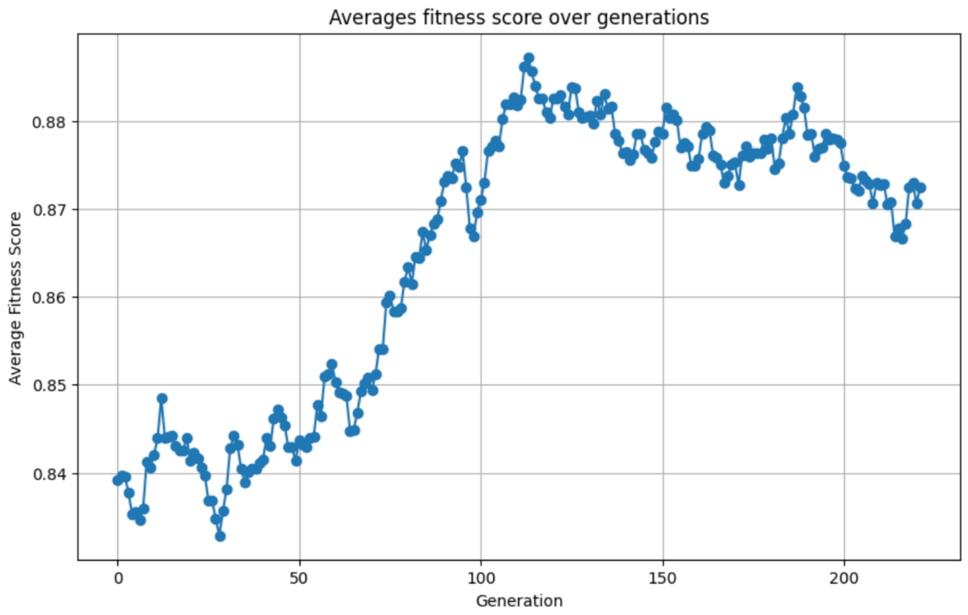
Convergence and Efficiency

The genetic algorithm demonstrated a rapid convergence rate, often reaching a near-optimal solution well before the maximum number of generations. This indicates its efficiency in exploring the solution space and avoiding local minima.



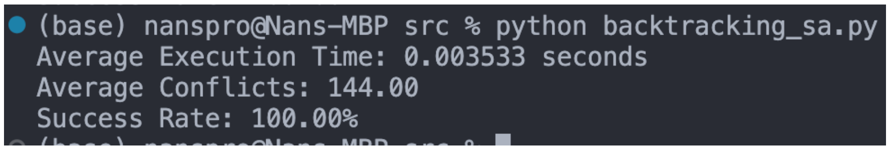
Performance Analysis

Gradual increase in average fitness score represents the algorithm getting better at finding solutions to a problem over time.



Backtracking Algorithm Results

Our focus was on scheduling constraint satisfaction problems, where we aimed to assess the effectiveness and efficiency of our backtracking algorithm enhanced with heuristics and advanced techniques. The time taken to find a solution, which reflects the efficiency of our method. The average execution time of the algorithm across 5 runs is less than 1 second per algorithm. Then, the feasible success rate, which represents the proportion of runs that found a feasible solution (even with conflicts), is 100%. This indicates the algorithm's ability to generate schedules that meet most constraints, albeit with some conflicts. Finally, the average number of conflicts per schedule was 144 Reducing this number is critical for creating more practical and error-free schedules.


```
● (base) nanspro@Nans-MBP src % python backtracking_sa.py
  Average Execution Time: 0.003533 seconds
  Average Conflicts: 144.00
  Success Rate: 100.00%
```

The analysis shows that while our algorithm is efficient in finding feasible solutions quickly, achieving conflict-free schedules remains a challenge. Further optimization and fine-tuning, especially in constraint handling and simulated annealing parameters, could improve the optimal success rate. These insights provide a clear direction for enhancing our scheduling algorithm to better meet the needs of our scheduling tasks.

Lessons Learned about AI topics:

Our project covers a wide array of AI topics, and undoubtedly, we have gained valuable insights and lessons throughout its development. Here's a breakdown of the lessons learned based on the specified topics:

Integrating Backtracking algorithms with CSPs:

1. Understanding Trade-offs: Integrating SA with CSP methods like backtracking, iterative refinements, and random starts has likely emphasized the importance of understanding trade-offs between exploration and exploitation. SA explores the solution space efficiently while avoiding local minima, whereas CSP methods offer systematic approaches to satisfying constraints.
2. Parameter Tuning: Learning to adjust SA parameters like initial temperature, cooling schedule, and neighborhood structure is crucial. This likely involved experimenting with different configurations to find the optimal balance between exploration and exploitation, considering the specific CSP constraints.

Optimization Techniques:

1. Multi-objective Optimization: Timetable generation inherently involves optimizing multiple objectives simultaneously, such as minimizing conflicts, maximizing room utilization, and satisfying hard constraints. Understanding how to balance conflicting objectives and trade-offs between them is a key lesson.
2. Performance Evaluation: Evaluating the performance of optimization algorithms involves more than just comparing solution quality. Metrics such as convergence speed, robustness to perturbations, and scalability likely became essential for gauging the effectiveness of the implemented techniques.

Heuristic Search Methods:

1. Exploration vs. Exploitation: Developing an intuition for when to explore new areas of the solution space (e.g., with SA) versus exploiting promising solutions (e.g., with GA) is crucial. This likely required understanding the characteristics of the problem space and the behavior of different search methods.

2. Diversification and Intensification: Recognizing the importance of balancing diversification (exploration) and intensification (exploitation) strategies within heuristic search algorithms. This balance ensures thorough exploration of the solution space while converging towards promising regions.

Constraint Satisfaction Problem:

1. Modeling Constraints: Translating real-world constraints into a formal CSP representation likely involved challenges in accurately capturing the complexity and interdependencies of scheduling constraints. Lessons learned likely include the importance of clear and comprehensive constraint modeling.
2. Constraint Propagation: Understanding how constraint propagation techniques, such as forward checking or arc consistency, can improve the efficiency of CSP solvers by pruning the search space based on partial assignments. Recognizing when and how to apply these techniques effectively is crucial.

Evolutionary Algorithms:

1. Population Dynamics: Understanding the role of population size, selection mechanisms, crossover, and mutation operators in the dynamics of evolutionary algorithms like GA. Experimentation likely revealed insights into how these factors influence convergence speed and solution diversity.
2. Adaptation and Diversity Maintenance: Learning to balance exploration and exploitation within evolutionary algorithms by adapting operator probabilities, mutation rates, and selection pressures. Maintaining diversity within the population is crucial for preventing premature convergence.

NP-Hard Problems:

1. Complexity Analysis: Developing an understanding of the computational complexity of scheduling problems and recognizing that timetable generation is often NP-hard. This likely involved exploring approximation algorithms, heuristic methods, and metaheuristic approaches to tackle the problem efficiently.
2. Algorithm Selection: Recognizing that no polynomial-time algorithm exists for solving NP-hard problems optimally, the team likely gained insights into the importance of algorithm selection based on problem characteristics, problem size, and available computational resources.

Overall, the project provided a rich learning experience across various AI topics, emphasizing the importance of algorithm design, parameter tuning, and problem modeling in tackling real-world optimization challenges.

Ethical AI Issues:

The use of AI algorithms like Simulated Annealing and CSP has shown promise in generating conflict-free schedules. However, there are several ethical considerations to keep in mind:

1. Fairness: Ensure that the scheduling algorithm does not favor certain professors or courses over others.
2. Transparency: Make the scheduling process transparent to all stakeholders, including professors, students, and administrators.

3. Privacy: Protect the privacy of professors and students by not sharing sensitive information in the schedule.
4. Bias: Be aware of any biases in the scheduling algorithm that may disadvantage certain groups or individuals.
5. Accountability: Ensure that the scheduling algorithm is accountable for its decisions and can be reviewed and audited if needed.

In summary, while AI algorithms can help optimize scheduling processes, it is essential to consider ethical implications and ensure that the scheduling system is fair, transparent, and accountable.

Improvements:

1. Soft Constraints: Incorporate soft constraints such as preferred time of class by professors, Preferred classroom by professors, Preferred time of class by student groups.
2. Complexity: The scheduling problem is inherently complex due to the large search space and numerous constraints. Future work could focus on refining the constraints and heuristics to improve scheduling efficiency.
3. Scalability: Scaling the scheduling algorithm to handle larger datasets and more complex constraints is a significant challenge. Future work could explore parallel processing and distributed computing to improve scalability.
4. Real-world Integration: Integrating the scheduling algorithm with real-world educational institutions poses challenges in terms of data integration, system compatibility, and user acceptance. Future work could involve pilot testing and user feedback to refine the scheduling system.
5. Optimization: Further optimization of the scheduling algorithm using advanced AI techniques such as deep learning and reinforcement learning could enhance scheduling efficiency and accuracy.

Conclusion:

The ETI (Educational Timetable Intelligent System) project has been instrumental in providing profound insights into the realm of artificial intelligence, specifically in scheduling optimization. Through the implementation of advanced algorithms such as Genetic Algorithms (GA), Simulated Annealing (SA), and Constraint Satisfaction Problem (CSP) techniques, our team has navigated the intricate challenges inherent in timetable generation.

Our journey has underscored the significance of striking a delicate balance between exploration and exploitation, tailoring algorithms to meet specific constraints, and optimizing the allocation of resources. By addressing these complexities head-on, our system represents a notable advancement in tackling NP-hard scheduling problems, demonstrating the transformative potential of AI in optimizing real-world processes.

In summary, the ETI project stands as a significant milestone in the field of AI-driven scheduling optimization. By harnessing a diverse array of algorithms and techniques, we have forged a resilient and effective scheduling system capable of crafting optimal timetables aligned with institutional requirements. As we reflect on our journey, we eagerly anticipate further strides in AI-driven optimization and the continuous refinement of our scheduling algorithms.

References:

1. Karova, M. N., Bojikova, V. T., & Mladenov, R. E. (2004). A Genetic Algorithm for a Student Timetabling Problem.
2. Herath, A. K. (2017). Genetic Algorithm For University Course Timetabling Problem.
3. Song, W., Cao, Z., Zhang, J., Xu, C., & Lim, A. (2022). Learning variable ordering heuristics for solving Constraint Satisfaction Problems. Engineering Applications of Artificial Intelligence, 109, 104603. https://doi.org/10.1016/j.engappai.2021.104603
4. Melício, F., Caldeira, P., & Rosa, A. (2000). Solving the Timetabling Problem with Simulated Annealing. In J. Filipe (Ed.), Enterprise Information Systems (pp. 171–178). Springer Netherlands. https://doi.org/10.1007/978-94-015-9518-6_18