Artificial Intelligence - Final Delivery

Optimization Methods/Meta-heuristics
Solving Google Hashcode 2021: Traffic Signaling

Given the description of a city plan and planned paths for all cars in that city, optimize the schedule of traffic lights to minimize the total amount of time spent in traffic, and help as many cars as possible reach their destination before a given deadline.

Grupo: 65_3A
Bruno Rosendo, up201906334
João Mesquita, up201906682
Rui Alves, up201905853

Problem Formulation

Given Input: Duration of the simulation; number of intersections; number of streets and, for each one, its name, time taken to cross it and identifiers to the intersections at the start and end of the street; number of cars and, for each one, the name of the streets it will traverse; the number of points given for each car reaching its destination before the simulation ends.

Solution Representation: Number of intersections and, for each one, its ID, the number of incoming streets covered by the scheduler and, for each street, the order and duration of the green lights.

Hard Constraint: For each second and intersection, only one car can travel to its next street, by going through a green light.

Evaluation Function: A value is given for each car that finishes its path before the end of the simulation. For a car that finishes its path at time T, it's given a value of X points (given by input) and an additionally (D - T) points, where D is the duration of the simulation.

Neighborhood/Mutation and Crossover Functions

Functions for neighborhoods and mutations:

- For each intersection and pair of green lights, swap their order.
- For each intersection and green light, randomly switch their order in the intersection.
- For each intersection and light, randomly increase/decrease its duration by a maximum of 10 seconds (if it becomes 0, the light will always be red).

Crossover functions:

- Uniform crossover, using a probability function to choose between Parent 1 and Parent 2 in each intersection, based on their fitness.
- Order-based crossover, choosing half of each intersection's lights of Parent 1 randomly, and filling the other half with the Parent 2's lights in order.

Implementation Work

- Programming Language: Python
- Development Environment: PyCharm / VSCode for development; Git and GitHub for version control.
- Data Structures:
  - Classes:
    - Car with id and array of streets, current street and remaining crossing time.
    - Intersection with id, outgoing streets, incoming streets, array with cycle of green semaphore, total time of the simulation and mutation/neighbourhood functions.
    - Street with id, name, starting intersection, ending intersection, time to cross the street, waiting queue and number of cars that pass the street.
    - Simulation with all cars, max time of simulation, bonus points for each car that concludes its path and evaluation function
    - Solution with intersections, a .Simulation and max execution time of that simulation; contains the algorithms used to solve the problem.
  - Configuration:
    - The algorithms and parameters used in the solution can be defined by the user. They are stored in a configuration dictionary, present in *config.py*.

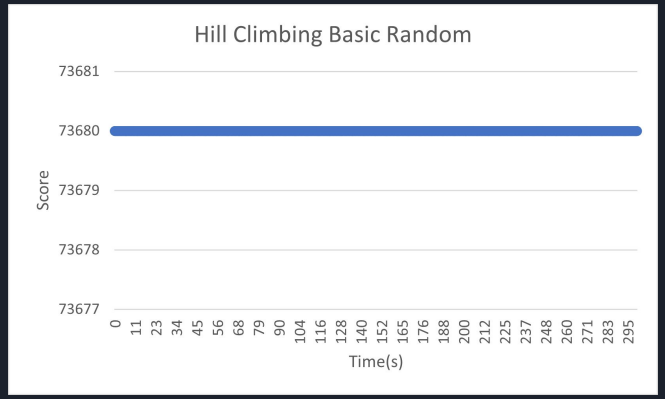
Approach

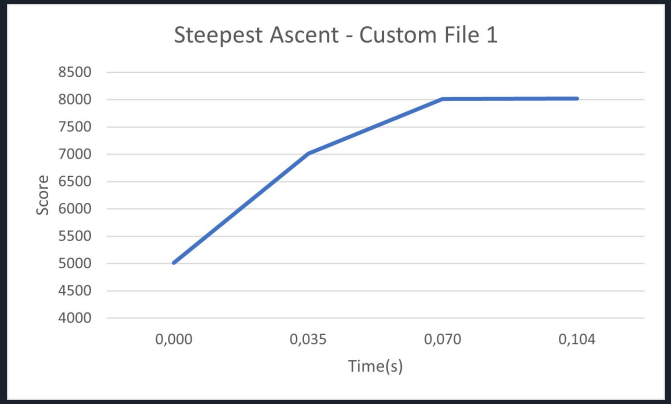
- In order to evaluate a solution, the simulation iterates every car for each second:
  - If the car is crossing a street, it decreases its crossing time.
  - If the car reached the end of its path's last street, it then removes the car from the simulation and increases the points of the solution.
  - If the car reached the end of the current street, it places the car in that street's queue.
  - If the car is at the end of a street and its semaphore is green, it then passes the intersection and goes to the next street in its path.
- Execution Flow:
  - The user configures the solution approach by input.
  - Remove streets that won't have any cars passing through it (optional).
  - Construction of an initial solution with basic green light cycle for each intersection, with 1 second duration for each semaphore in the intersection.
  - Run the algorithm that the user selected, with the parameters according to its chosen configuration.

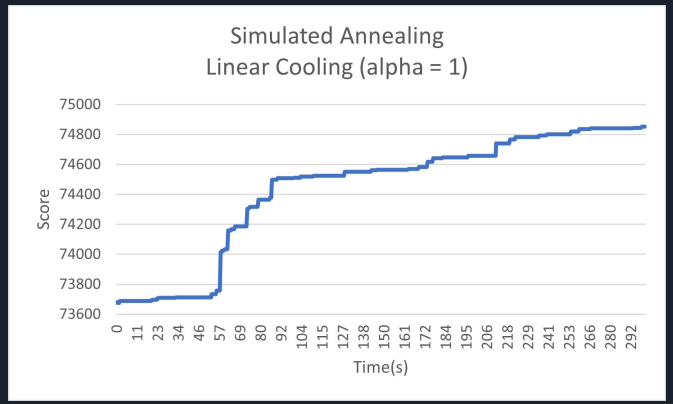
Implemented Algorithms

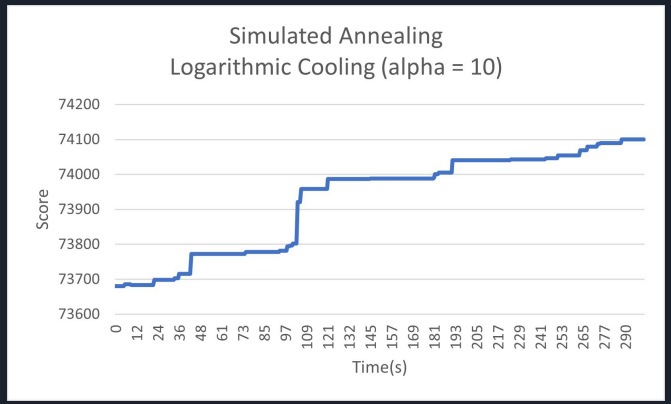
- Local Search:
  - **Basic Hill Climbing (random):** Generates a random neighbour of the current solution and accepts it if it's better evaluated than the current solution.
  - **Steepest Ascent:** Generates all possible neighbours and selects the best evaluated one.
- Simulated Annealing
  - Starts with a given solution and temperature, chooses a random neighbour and accepts it if it's better evaluated than the current solution, otherwise only accepts it with a probability that depends on the temperature.
- Tabu Search:
  - Starts with a given solution and collects some of its neighbours. It penalizes moves that take the solution into a previously visited neighbour, by keeping a Tabu List that stores those neighbours for a number of iterations (Tabu Tenure). However, it accepts non-improving solutions in order to prevent getting stuck in local minimums and explore different neighborhoods.
- Genetic algorithm:
  - Inspired by the theory of natural selection, relies on biological operators such as mutation (given probability), crossover (uniform or order-based) and selection (random or roulette).
  - **Generational GA:** Replaces the whole population, in each generation.
  - **Steady GA:** Replaces only the worst individual in the population, in each generation.

Experimental Results (Custom File 2)

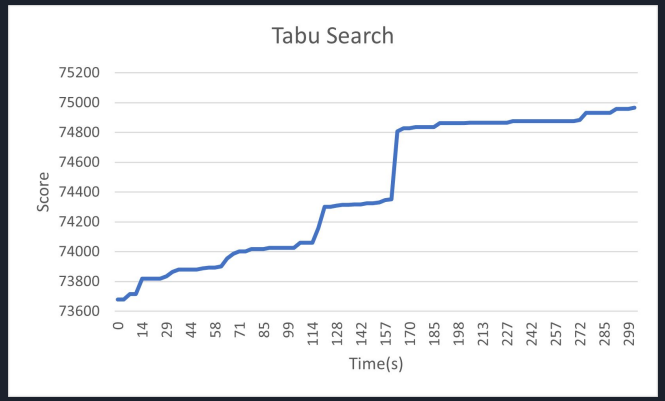


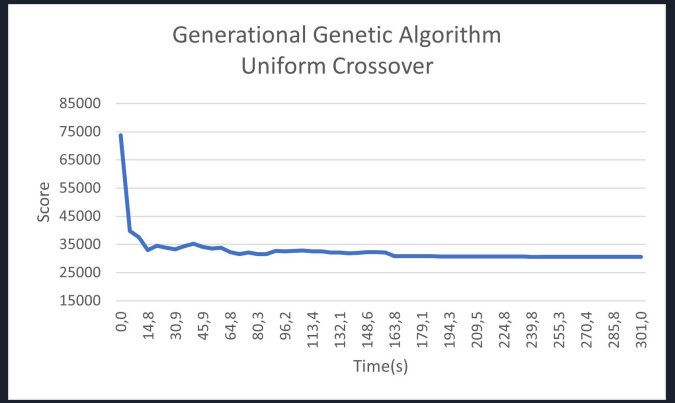


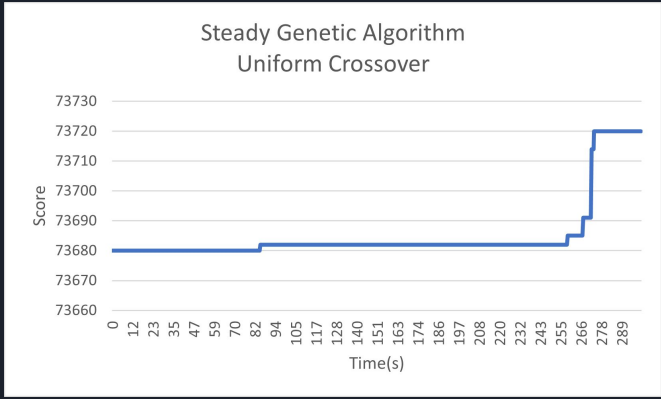


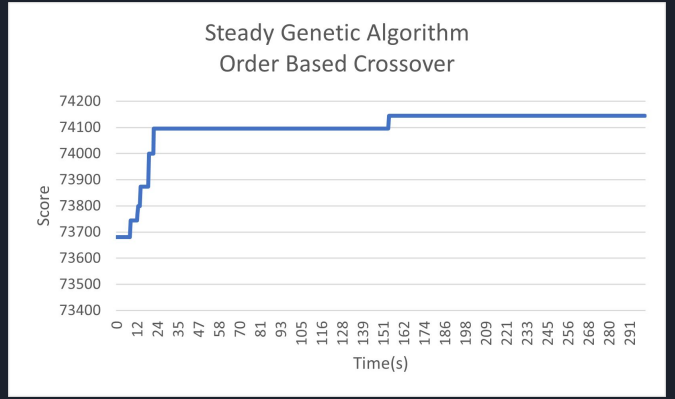


Experimental Results (Custom File 2)









Conclusions

The greatest challenge of this project was making an efficient evaluation function, given the nature of the problem and the dimensions of the input files, which required heavy computational tasks. As a consequence, the algorithms that utilize the evaluation function more often will take longer, resulting in a premature solution.

Looking at the graphs, the *Simulated Annealing* with linear cooling and the *Tabu Search* yielded the best results. On the other hand, some of them were not as successful:

- **Basic Hill Climbing** - The algorithm is inevitably stuck, since the initial solution is itself a local maximum.
- **Steepest Ascent** - The algorithm is incapable of running the larger input files within a feasible time frame, since it calculates the evaluation of all the possible neighbours in each iteration. For this reason, the result presented was measured using a simpler test case. Additionally, it'd also have the same problem as above.
- **Generational Genetic** - The algorithm is not able to find better solutions using diverse crossover functions, such as *Uniform* and *Order-Based* crossover. This is due to the problem we're working with, which favors solutions based on neighborhood functions.

Work References

- Hash Code 2021
- Sagishporer - Hashcode 2021 Qualification
- A Comparison of a Genetic Algorithm and Simulated Annealing Applied to a Traffic Light Control Problem
- Hashcode 2021: A lucky ride
- Genetic Algorithms - Crossover
- A Comparison of Cooling Schedules for Simulated Annealing
- Tabu Search
- Google Hashcode 2021 Score Calculator