Self Driving Rides
Google Hashcode 2018

Artificial Intelligence 2019/2020

Diogo Machado	-	up201706832
Gonçalo Marantes	-	up201706917
Leonardo Moura	-	up201706907

Problem Specification

Problem Statement: Given a list of pre-booked rides in a city and a fleet of self-driving vehicles, assign rides to vehicles so that riders get to their destination on time.

Objective Function: The solution score is calculated by the sum of each completed ride’s distance with the ride’s bonus if bonus conditions were met.

Optimization Problem

- **Initial Solution:** Greedy approach, so as to start with a solution that is close to optimal;
- **Constraints:**
  - For each vehicle, *totalSteps* $\leq$ *T*;
  - Each ride can be associated to, at most, one vehicle;
  - There can be no incompatible rides on one same vehicle;
- **Assessment Function:** Sum of each vehicle's score. Vehicle's score is calculated by: *vehicle[i] * (rides[i].score + rides[i].bonus)*

Hill Climbing / Simulated Annealing:

- **Representation:** Each vehicle is represented by a binary list, with length equal to the number of rides, and where the bits that are set are the rides associated to the vehicle;
- **Neighbour selection:** For each ride, attempt to associate it to a different vehicle - if the result is valid, we have one of the neighbours.

Genetic Algorithms:

- **Representation:** Binary matrix, each row represents a vehicle, each column represents a ride;
- **Mutation:** Random bit inversion;
- **Crossover:** For each row, single point crossover

Project Structure

- **Development Environment:** PyCharm
- **Programming Language:** Python 3
- **Data Structures:** OOP Approach, Binary Lists and Matrices
    - Dataset Class
    - Ride Class
    - Car Class
    - Solution Class
    - Algorithms: Hill Climbing Class, Simulated Annealing Class, Steepest Ascent Class, Genetic Class
- **File Structure:**
    - inputs/ -> Contains the input files from the competition
    - src/ -> Contains the source code for the implemented algorithms

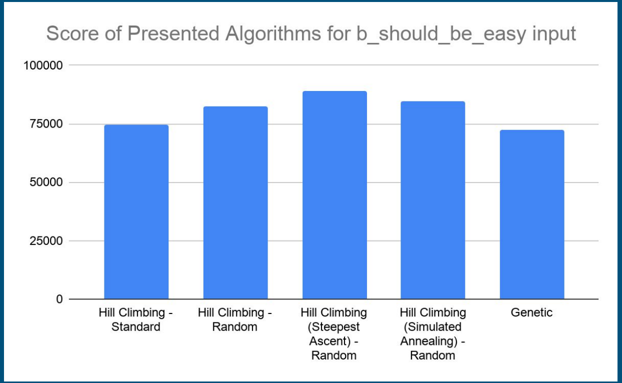
Approach

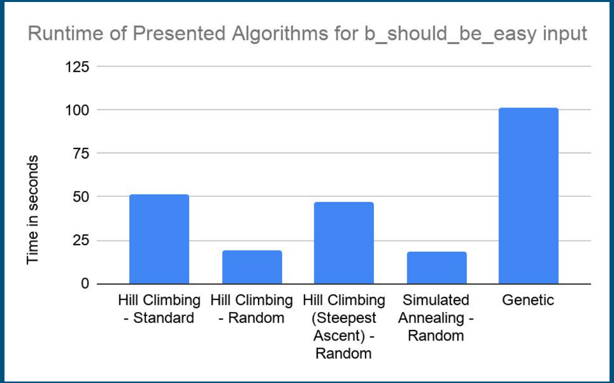
- **Initial greedy or random solution:** In order to have a starting point from where to start applying the proposed algorithms, a greedy solution was elaborated.
  - Note: we observed that solving the problems from scratch, instead of starting with a greedy solution, yielded similar results.
- **Objective Function:** The score of each solution in the context of the competition. Each ride completed before its *latest finish* is worth the distance traveled in that ride. Each ride has an associated bonus if the ride starts on its *earliest start*. The final score is the sum of the scores of all allocated rides plus any applicable bonuses. This is the value we want to maximize.
- **Heuristics:** Some of the aspects of our problem required the employment of heuristics, in order to quickly generate better solutions. One of them was the ordering of the rides for each car, based on their earliest starting time and total distance.

Implemented Algorithms

- **Classic Hill Climbing:** Generates one neighbor at a time, representing a possible solution. When a neighbour with a better score than the current one is found, that solution is applied and we start searching for neighbors using that solution. This process is repeated, converging for an optimal solution. The random version of the algorithm is also implemented. A random neighbor is generated and and selected for the next iteration.
- **Steepest Ascent Hill Climbing:** A variation of the Hill Climbing algorithm. All the possible neighbors of a solution are generated and the closest one to the solution is selected.
- **Simulated Annealing:** This algorithm aims to solve the problem Hill Climbing has with local extrema. It does this by allowing some “bad” moves while decreasing their frequency and intensity.
- **Genetic algorithms:** These algorithms simulate natural selection by starting with a base population and applying a crossover function with some probability of mutation to the members of that population, creating a new generation. A fitness function is used to evaluate the members of the new generation and select the fittest ones. This way the population will evolve, converging for an optimal solution.

Results - *b_should_be_easy*

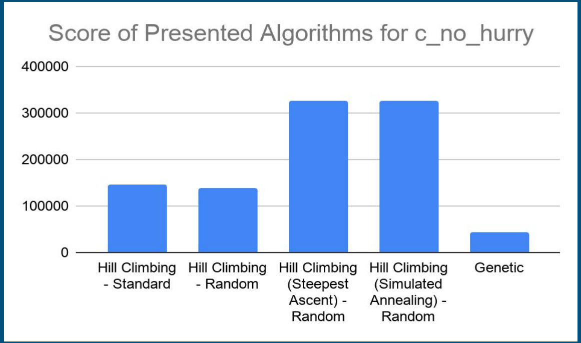


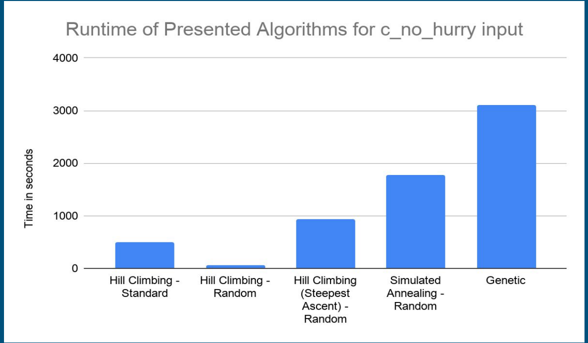


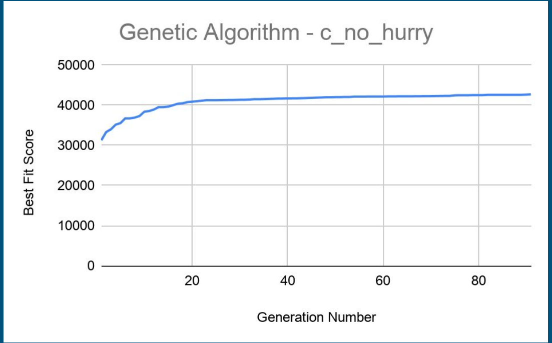
Score of Presented Algorithms Input *c_no_hurry*	**Score**
Hill Climbing - Standard	74728
Hill Climbing - Random	82529
Hill Climbing (Steepest Ascent) - Random	88811
Simulated Annealing - Random	84694
Genetic	72332

Runtime of Presented Algorithms Input *c_no_hurry*	**Time (seconds)**
Hill Climbing - Standard	51.74
Hill Climbing - Random	19.38
Hill Climbing (Steepest Ascent) - Random	46.7
Simulated Annealing - Random	18.5
Genetic	101.51

Results - *c_no_hurry*







Score of Presented Algorithms Input *c_no_hurry*	Score
Hill Climbing - Standard	145150
Hill Climbing - Random	137610
Hill Climbing (Steepest Ascent) - Random	326519
Simulated Annealing - Random	326450
Genetic	42705

Runtime of Presented Algorithms Input *c_no_hurry*	Time (seconds)
Hill Climbing - Standard	491.95
Hill Climbing - Random	51.69
Hill Climbing (Steepest Ascent) - Random	928.58
Simulated Annealing - Random	1774.6
Genetic	3109

Generation Number Input *c_no_hurry*	Score
Generation #1	31276
Generation #45	41824
Generation #91	42702

Conclusions

- During the elaboration of this project, we were able to verify the effectiveness of the applied algorithms. Even though they aren't as fast as a possible greedy approach, they guarantee superior results, if properly structured;
- Which brings us to the second conclusion: proper structuring and modelling of the problem and all its context and specifications is key for a successful implementation of a solution for it using metaheuristics (in our case, Hill Climbing variants and Genetic Algorithms);
- Small optimizations matter - when dealing with large amounts of data, even the smallest optimization can have a very tangible effect on the performance and outcome of an algorithm;
- Trial and Error is only bad if done aimlessly - with proper organization, trial and error is a valid way of solving complex problems (especially NP-Complete problems, which are not easily solved without the use of this approach).

References

- Stuart, R., 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall;
- IART Slides on Solving Search Problems;
- IART Slides on Optimization and Genetic Algorithms;
- Problem Statement;
- Python 3 Documentation;
- Python Algorithms;
- Priority-based Approach for the Problem;
- Greedy Approach to the Problem;
- Study of the appropriateness of Genetic Algorithms to the Problem;
- Genetic Algorithm approach to the Problem;
- Mixed 0-1 Integer Programming (MIP) Approach to the Problem;