U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

IA Project
Traffic signaling

Grupo 66_3A

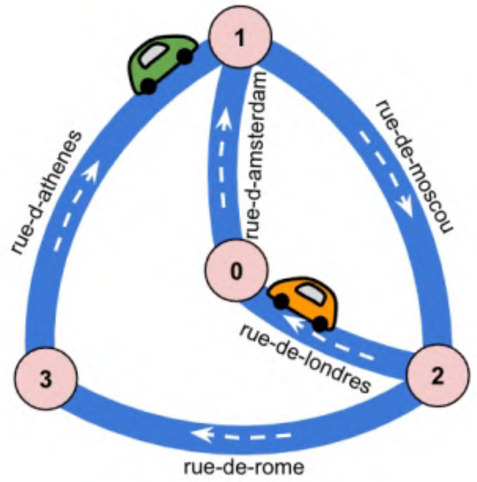
Francisco Pires up201908044

Sérgio da Gama up201906690

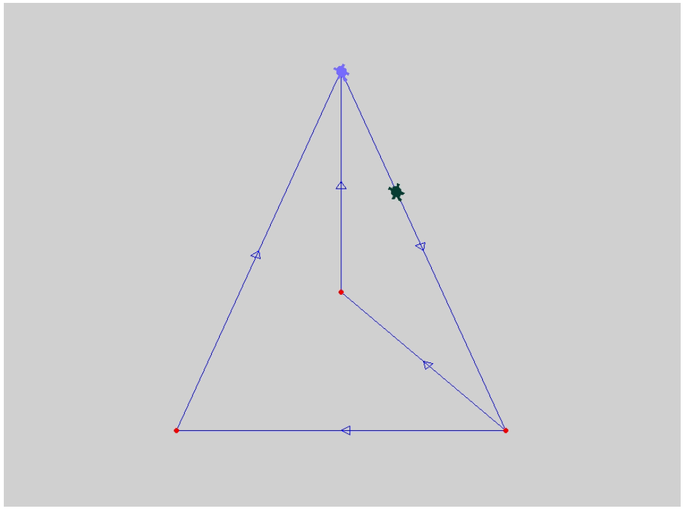
Pedro Nunes up201905396

Specification of the assignment

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.



our approach in representing a city plan



The state space can be calculated based on the following formula, whereas

TotalInt - Total of intersections

numSem - Total of semaphores of the nth intersection

totalSeconds - Max seconds the semaphores can hold

$$StateSpace = \sum_{n=0}^{TotalInt} \left(numSem_n!\right)^{totalSeconds}$$

Optimization problem formulation

In order to solve the optimization problem it was necessary to be able to simulate a given state (or solution). Therefore the first step was to define said state and structure how to run the simulation according to it.

Our simulation state is mainly composed by a list of intersections, that inside them have a list of traffic lights. Each of the traffic lights is assigned to one and only one street, a green light time (seconds) and has a queue of cars.

Abstract view of the simulation state

```python
class Intersection:
    def __init__(self, id):
        self.id = id
        self.traffic_lights = []
        self.current_light = 0
        self.num_of_lights = 0
```

```python
class TrafficLight:
    def __init__(self, street, time):
        self.cars_queue = queue.Queue()
        self.time = time
        self.street = street
        self.current_time = time
```

```python
class Car:
    def __init__(self, path_length, path):
        self.path_length = path_length
        self.path = path
        self.current_street = 0
        self.remaining_cost = 1
        self.draw_on = True
```

Main simulation objects classes

Solving the optimization problem

The next step in solving this problem was to implement various optimization algorithms and see which ones gave better results.

✓ **Hill climbing** Local search

✓ **Simulated Annealing** Global optimization

✓ **Genetic Algorithm** Global optimization

Notes

- How can we reach the states space?
- Should we go for a faster solution or a better solution?

Optimization problem implementation

What did our algorithms have in common?

Score evaluation

This function is responsible for evaluating the simulation states, by attributing them a score. It will be used to compare states and see which are better.

- For each car that reaches the end of their path within the time limit, the score increases **M** points as specified by the city plan
- If a car reaches **n** seconds sooner, the score further increases **n** points
- Used in all the algorithms

Insert Neighbour function

We needed to be able to change the traffic lights green time duration, thus opening up the need to create a neighbour function.

- Randomly chooses an intersection and gives a duration (between **1** and **total simulation time**) to a random traffic light
- Used in both simulated annealing and hill climbing
- Makes us reach most of the state space

Swap Neighbour function

The traffic lights were changed in a cyclic behavior, thus we needed a way to change the order of the cycle.

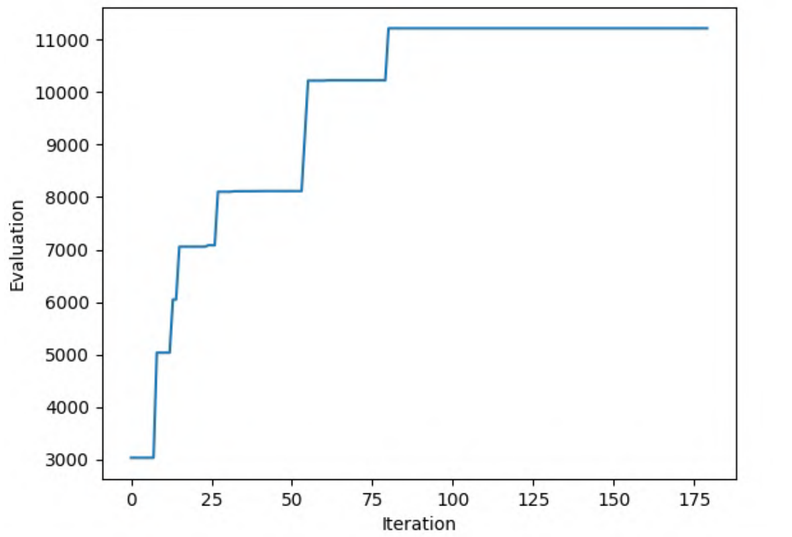
- Randomly chooses an intersection and swaps between **two random** traffic lights.
- Helps us reach more of the states space

Hill Climbing

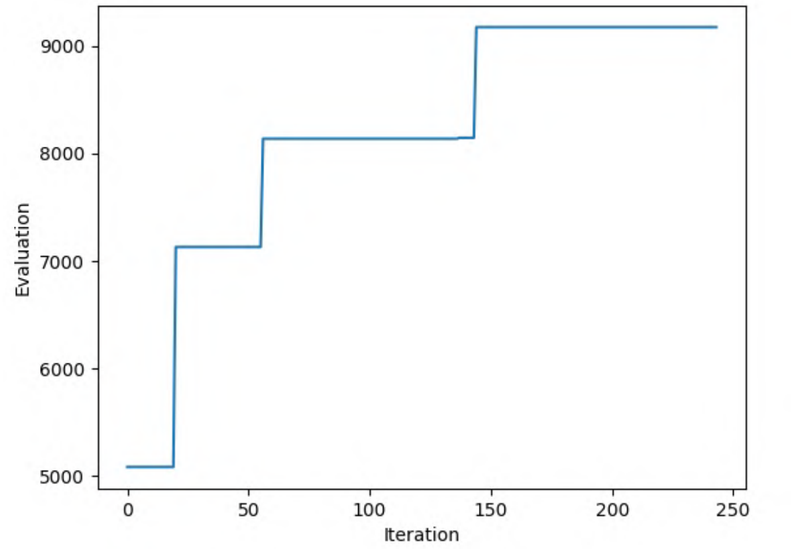
Local search algorithm that starts with a random solution and then attempts to find a better solution by making changes to it.

How does the algorithm works?

- Two different versions were implemented, **random** and **steepest ascent**.
- The **random version** randomly chooses a neighbour function and generates a state from it. If its better we take it, if not we continue to generate states at random .
- The **steepest ascent** version generates **all** possible neighbours and chooses the best of them all.
- The algorithm stops when no better solution is found after **n** iterations.



Score Evaluation of steepest ascent version



Score Evolution of random version

Simulated Annealing

Global optimization algorithm that mimics the process of slow cooling of metals

How does the algorithm works?

- The algorithm iterates over a fixed number of iterations and in each iteration:
  - Calculates the temperature value, based on a given cooling schedule
  - Chooses a random neighbour state
  - Accepts the chosen state if it has a better score
  - If it doesn't have a better score, it accepts the new state with a probability given by this formula (difference between scores divided by the current temperature): $e^{\Delta E/T}$
- This ensures that the algorithm doesn't stays stuck in a local minimum

Cooling Schedules

The implementation of this algorithm requires a generation of a finite sequence of decreasing values of temperature (T). To achieve this, it is used a cooling shcedule function.

Multiplicative Monotonic Cooling:

- **T** is calculated by multiplying the initial temperature by a factor (Exponential, Logarithmic, Linear or Quadratic) that **decreases** with respect to iteration **i**

Additive Monotonic Cooling:

- It has 2 additional parameters, final temperature (**tn**) and number of iterations. A factor (Linear or Quadratic) that that **decreases** with respect to iteration **i** is subtracted to **tn**

Non-Monotonic Adaptive Cooling:

- This multiplies one of the former criteria resultant **T** by the difference between the current state and the best state so far. That way it **adapts** to the current state of the search problem

Simulated Annealing

Results

All the following tests were made using the *small_map.txt* city map, with number of iterations set to **100** and the initial temperature set to **100**. Moreover, it was used always the same neighbour function, that changed randomly the traffic lights green time.

1. Exponential multiplicative cooling

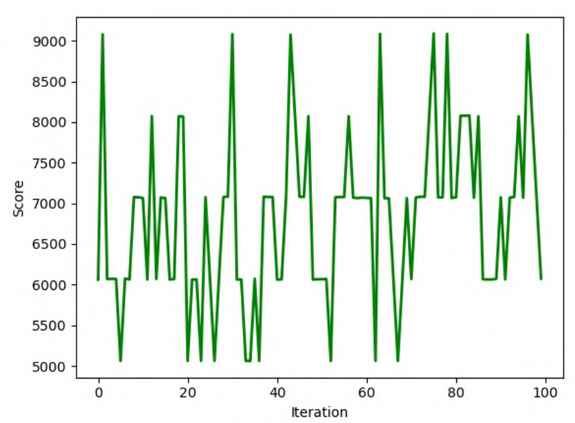
Exponential monotonic cooling schedule decreases the temperature very fast, which makes the algorithm stop in a local minimum. This is visible in the graph on the left, were the final value (best score) **9090** is achieved multiple times through the execution of the algorithm.

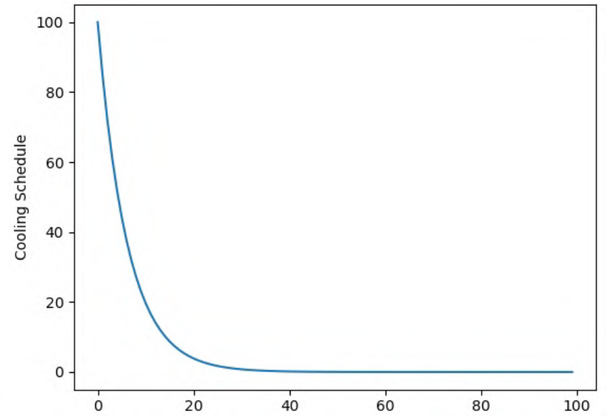
2. Linear additive cooling

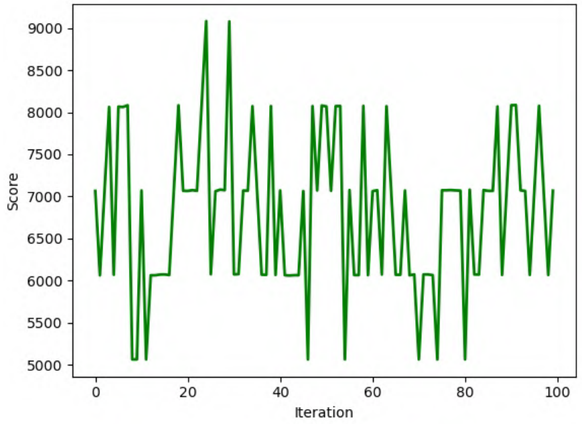
In contrary to the last schedule, this one decreases a lot slower (linearly instead of exponentially). Because of that, this presents a slower convergence, only reaching **9084** in the end of the 100 iterations.

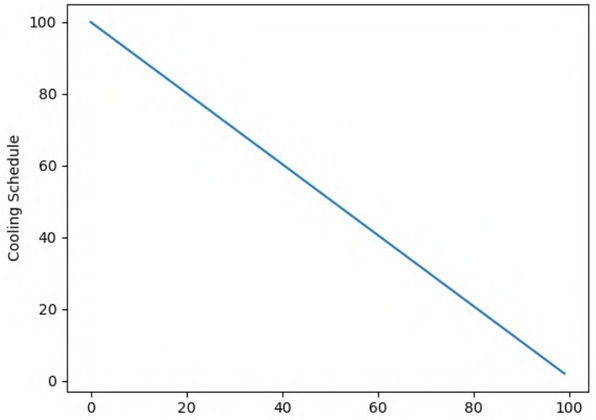
3. Linear additive **adaptive** cooling

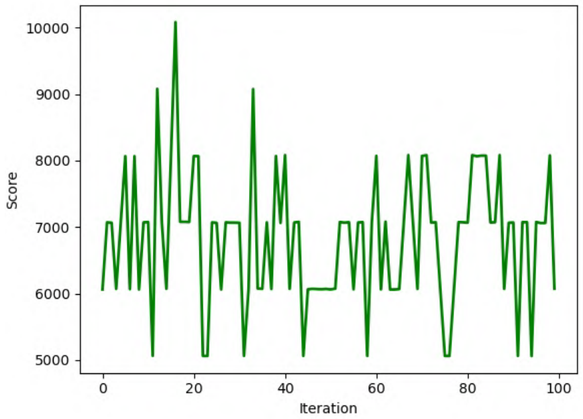
Lastly, using the adaptive cooling over the linear additive schedule we are able to achieve a better result, because of the improved ability to escape from worse solutions states. The best score achieved was then **1084**.

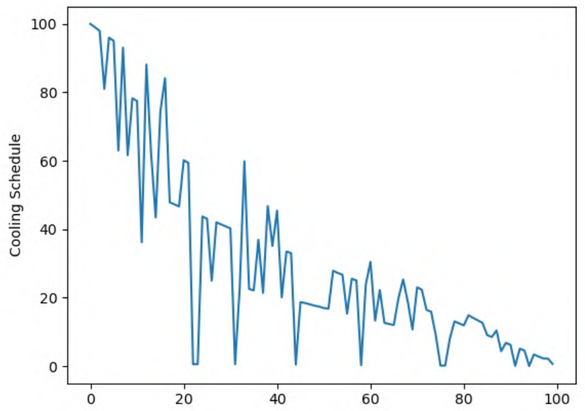












Score Evolution

Cooling Schedules

Genetic Algorithm

Based on the mechanics of biological evolution
The following generations tend to get better as they descend from the fittest individuals of the previous generations
In this implementation we control execution through 3 values: **P**, **G** and **M**

How does the algorithm works?

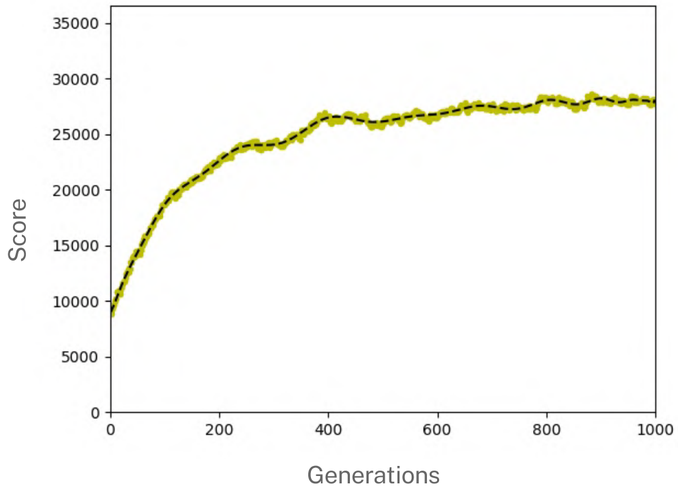
- Based on the mechanics of biological evolution
- Initialization: Start with a population of **P** random solutions
- Evaluation: calculate fitness based on the scores produced
- Crossover: Pick pairs of parents from the population based on their fitness and generate a new generation of size **P** of solutions by incorporating parts of each parents solution
- Mutation: For each individual in the population and given a probability **M** of mutation, randomly change its state
- Repeat for **G** generations and return the best performing individual

Crossover

The child's state keeps the order of traffic lights within each intersection the same as the first parent but the values of time of each traffic light are equal from both parents

Mutation

When applying mutation we look at every intersection in the state and randomly, with a probability **M**, swap the order of any two traffic lights within. Also each traffic light time value may be changed by +/- 1 second ensuring it stays within the interval [0, **T**], with **T** being the total execution time for the defined city plan



Average Generation Score and its Regression

Results analysis

Hill Climbing steepest ascent

The first graphic displays the **best** scores from running hill-climbing **100 times with 100 iterations each**. Every iterations was performed with a **random** initial state. Although it was pretty fast to compute,the majority of our solutions belong in the **[10000,14000]** points.

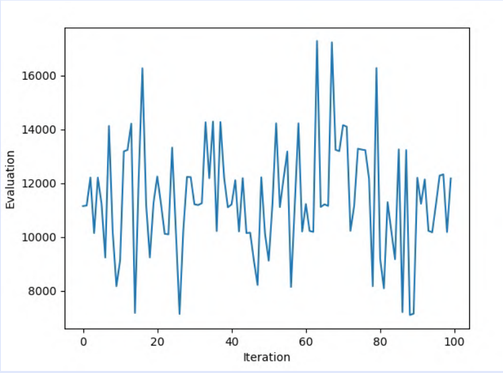
Simulated Annealing (com metodo tal e tal)

The simulated annealing graphic was obtained by running the algorithm using a linear schedule, with initial temperature set to **100**, with **100** iterations, achieving a best score of **16333**. Therefore it did not reached a result as good as the genetic algorithm, because of the slow convergence schedule and the few number of iterations.

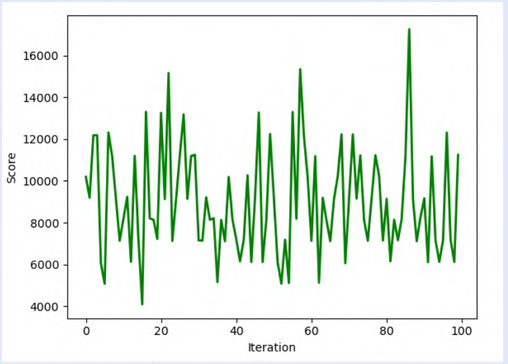
Genetic Algorithm

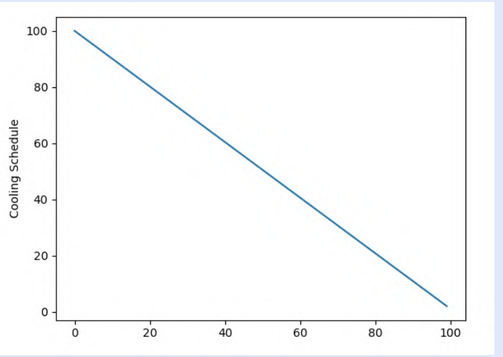
Lastly we ran the genetic algorithm for 100 generations and a population of 100 with 10% mutation probability. On the left graph we can see each generation best score reaching at best somewhere around **21000** a significantly better result than the others albeit only after a large number of iterations. On the right we can see the average score of each generation where it is apparent that each generation comes closer to better solution neighbourhood and indicates that in further generations it might reach even better solutions.

Hill Climbing

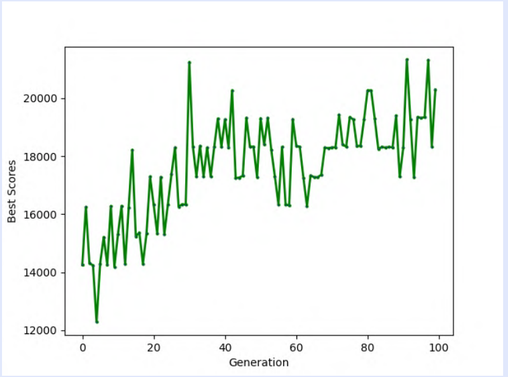


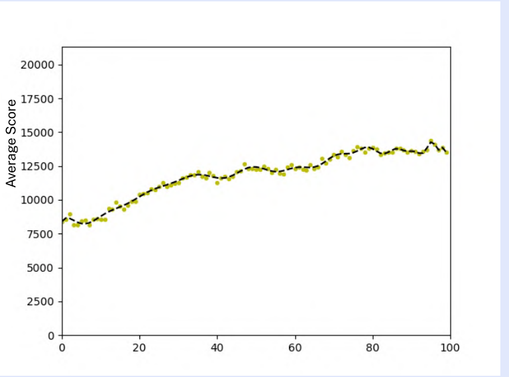
Simulated Annealing





Genetic Algorithm





Comparison between the algorithms

Conclusion

In **100 runs** Hill-Climbing only got a score **above 16000 four times** whereas the other approaches easily had it with only one run. We can conclude that it always got stuck in local maximums which shows its heavy dependency on the initial solution.

The simulated annealing ability to reach a good result heavily **depends on the number of iterations** and the rest of the parameters given, like the **initial and final temperature**. Moreover, the cooling schedule also impacts the algorithm performance. With slow convergence schedules, a good result takes more iterations and time to accomplish. Oh the other hand, fast convergence schedules are faster and are more likely to be stuck in a local maximum.

The genetic algorithm seems to be the most promising as it usually reaches significantly better results, although it **needs a sizeable population** for each generation for it to be really effective, which is a lot more computationally demanding. In spite of this it presents the advantage of slowly improving the overall score of the individuals of the population which means that it can see improvements with the increase in iterations for longer than the other algorithms.

References

- http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/
- Artificial Intelligence A Modern Approach by Stuart Russel and Peter Norvig
- Moodle slides