U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Drone Delivery

Google Hash Code, G20 IART 2020/2021

André Gomes up201806224 Gonçalo Teixeira up201806562 Filipe Recharte up201806743

Input File

- Map
- Products
- Warehouses
- Orders
- Drones
- Commands
  - Load
  - Deliver

100 100 3 50 500	100 rows, 100 columns, 3 drones, 50 turns, max payload is 500u
3	There are 3 different product types.
100 5 450	The product types weigh: 100u, 5u, 450u.
2	There are 2 warehouses.
0 0	First warehouse is located at [0, 0].
5 1 0	It stores 5 items of product 0 and 1 of product 1.
5 5	Second warehouse is located at [5, 5].
0 10 2	It stores 10 items of product 1 and 2 items of product 2.
3	There are 3 orders.
1 1	First order to be delivered to [1, 1].
2	First order contains 2 items.
2 0	Items of product types: 2, 0.
3 3	Second order to be delivered to [3, 3].
1	Second order contains 1 item.
0	Items of product types: 0
5 6	Third order to be delivered to [5, 6]
1	Third order contains 1 item.
2	Items of product types: 2.

Fig1 – Example input file

Solution Representation

Vehicle: If value is None, no drone is assigned to that gene, but it can be used for future generations. If the value is an integer, then a drone is assigned to the task represented by the gene.

Demand: weight of the product to be delivered, if the value is negative, or picked up, if positive

Node: Identifier of a WH or a order. The order is identified and not the delivery point because multiple orders can have the same delivery point

Product: Identifier of the product

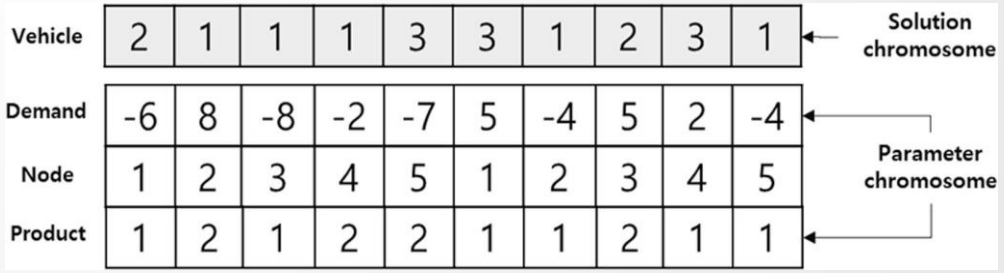
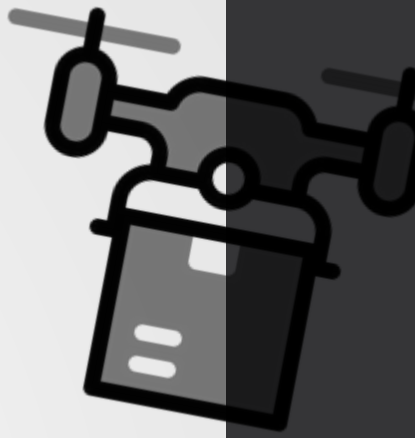


Fig2 - Visual representation of a chromosome

Neighbourhood/Mutation



Pq ➡ Product quantity

Dc ➡ Drone capacity

Cromossome Mutations

- **Alter trajectories** by switching drones of 2 genes
- **Unbalance the quantities** of 2 alleles of the same WH with the same item
- **Add Supply genes**: If there are items in WH's that are not beeing picked up, adds a gene with a random quantity of that item, between [1, min(Pq,Dc)]
- **Join 2 genes:** They can be of type deliver or supply, but need to have the same node. After joining, the resulting gene can stay on the position of the first gene or the second
- **Remove one or all genes with penalty**

Gene Mutations

- **Split a gene** of type supplier or deliver into two genes, of either balanced or unbalanced quantities
- **Alter Associated Drone** by replacing the current drone identifier with another or by 0

Crossover functions

Cause changes to the drones trajectories

- A drone that picks up products can deliver it to a different client if it is more beneficial.

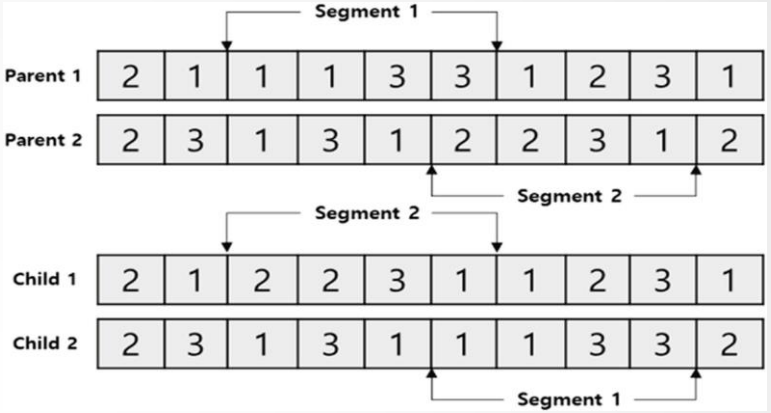


Fig3 - Visual representation of a crossover

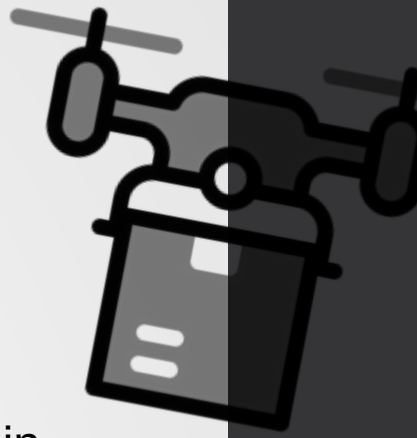
Rigid Constraints

- Drone Route cannot surpass the defined number of turns
- Drone cannot carry more weight than it’s capacity and cannot carry negative weight
- Drone can only deliver products that he is currently carriyng

Evaluation functions

- We calculate the score of an order with the following formula: **(T − t) / T × 100,** where **t** is the turn when an order is completed, and **T** is the total turns of the simulation. Adding all the scores, we obtain the solution (Chromosome) score.

Implementation



Classes

List of python classes to represent the main objects of the problem:

- Spot(ABC)
- Warehouse(Spot)
- Order(Spot)
- Product
- DronePath
- OrderPath
- Shipment
- Problem
- Point
- Chromosome
- Gene

Data Structures

- **Lists**, for representing the Genes in a Chromossome
- **Dictionaries**, for representing the products and respective quantities
- **Tuples**

Algorithms

- Greedy (best solution)
- Greedy (random solution)
- Naive

Metaheuristics

- Hill Climbing
- Genetic
- Simulated Annealing

Experimental Results

File: demo_altered (3 drones, 3 product types, 2 warehouses, 3 orders)

Iterations: 200

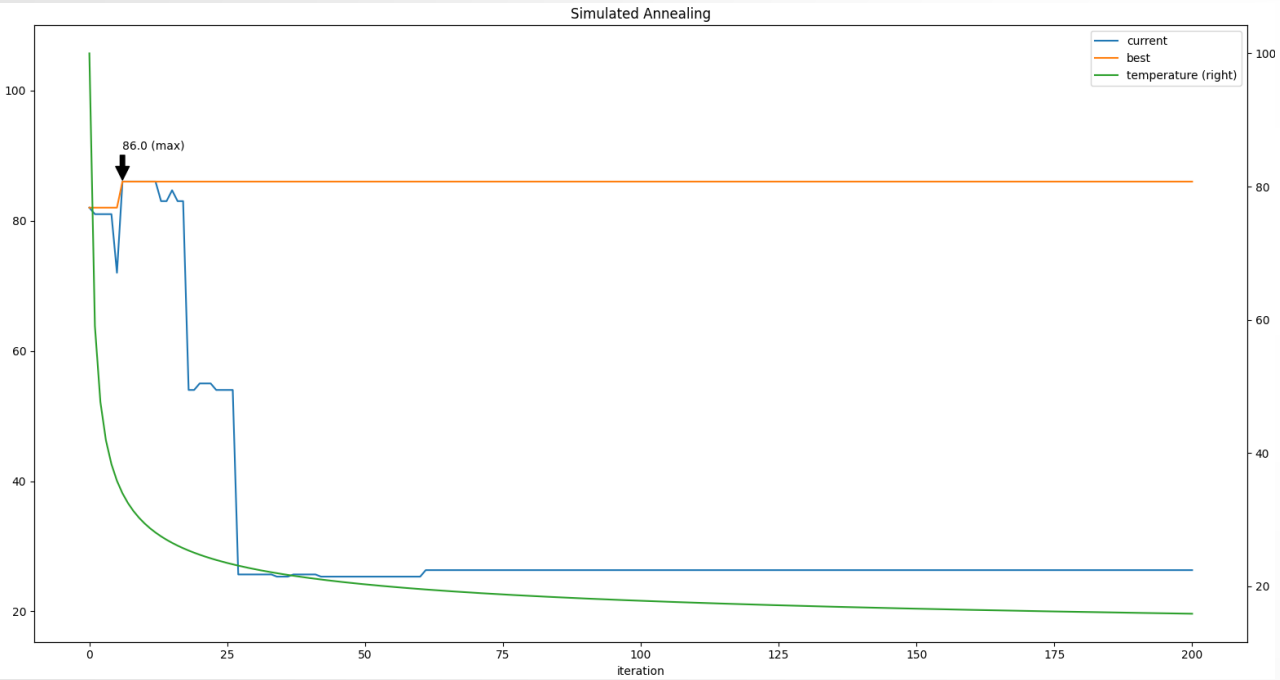


Fig4 – **Simulated annealing graphic**, starting with a solution with score 83, and ending with 86.0

Iterations: 20 Simulated Annealing iterations – with 100 iterations each one

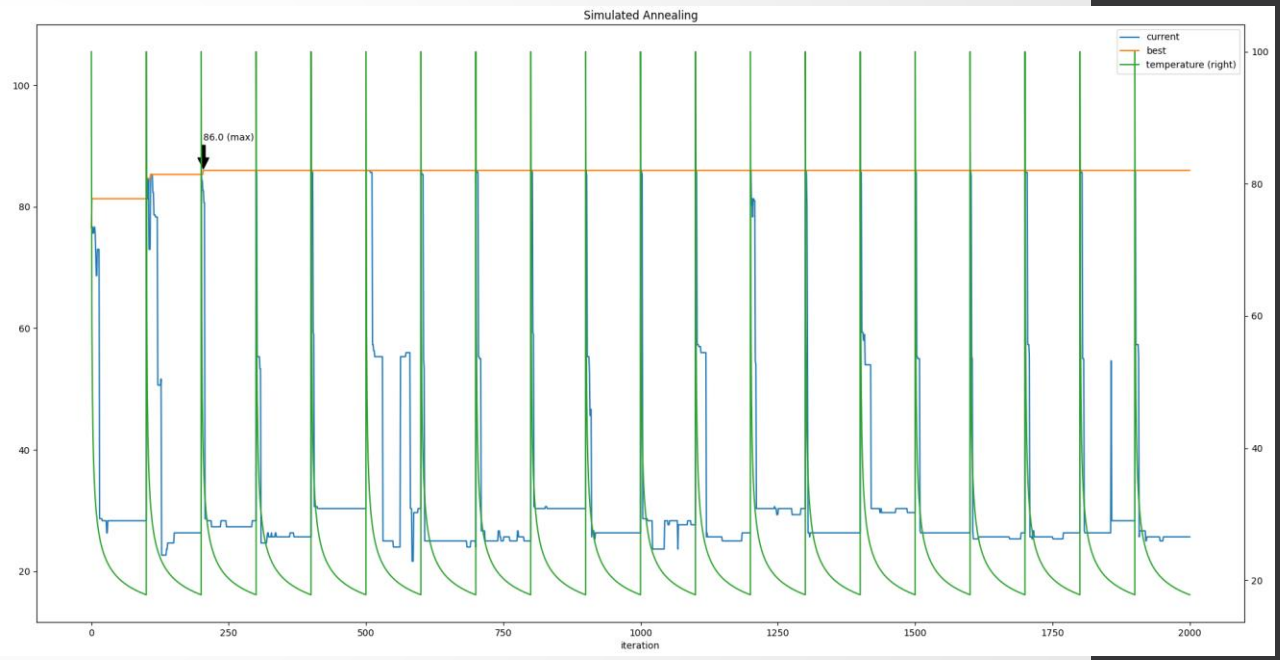


Fig5 – **Iterative Simulated Annealing** graphic, starting with a solution with score 83, and ending with 86.0

Experimental Results

File: demo_altered (3 drones, 3 product types, 2 warehouses, 3 orders)

Mutation rate: 0.85

Crossover rate: 0.85

Initial population: 200

Generations: 2000

Iterations: 500

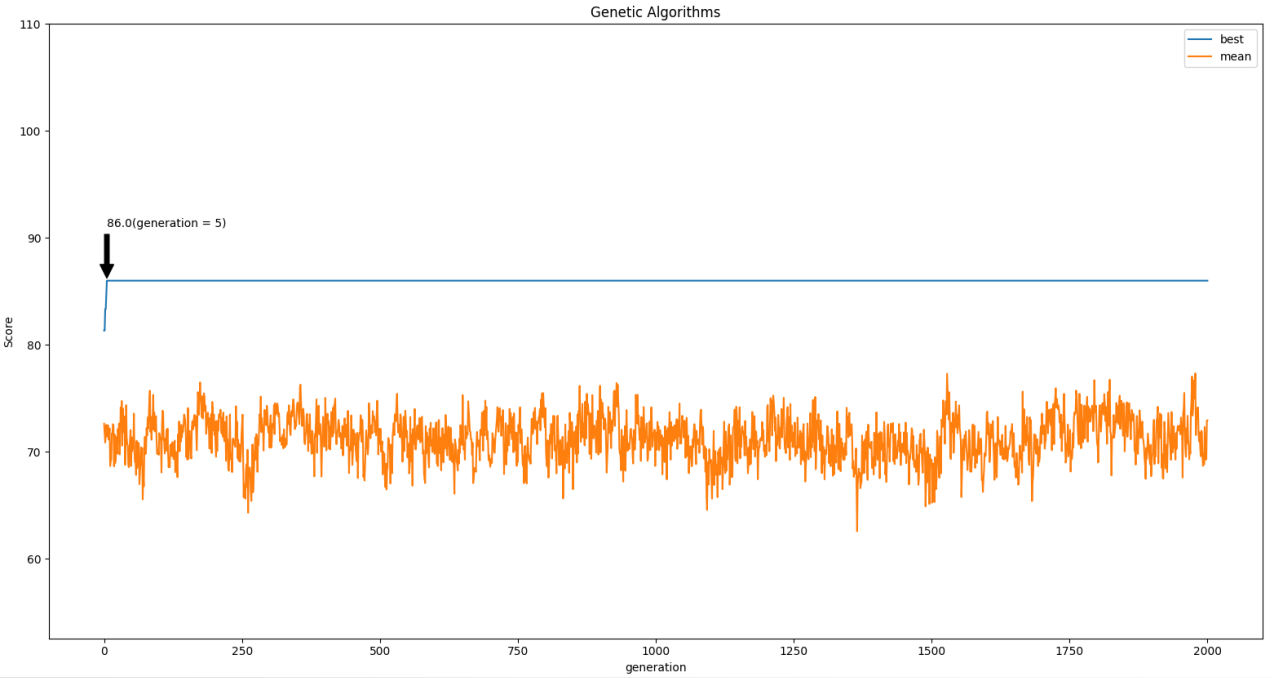


Fig6 – **Genetic algorithm graphic**, starting with a solution with score 83, and ending with 86.0

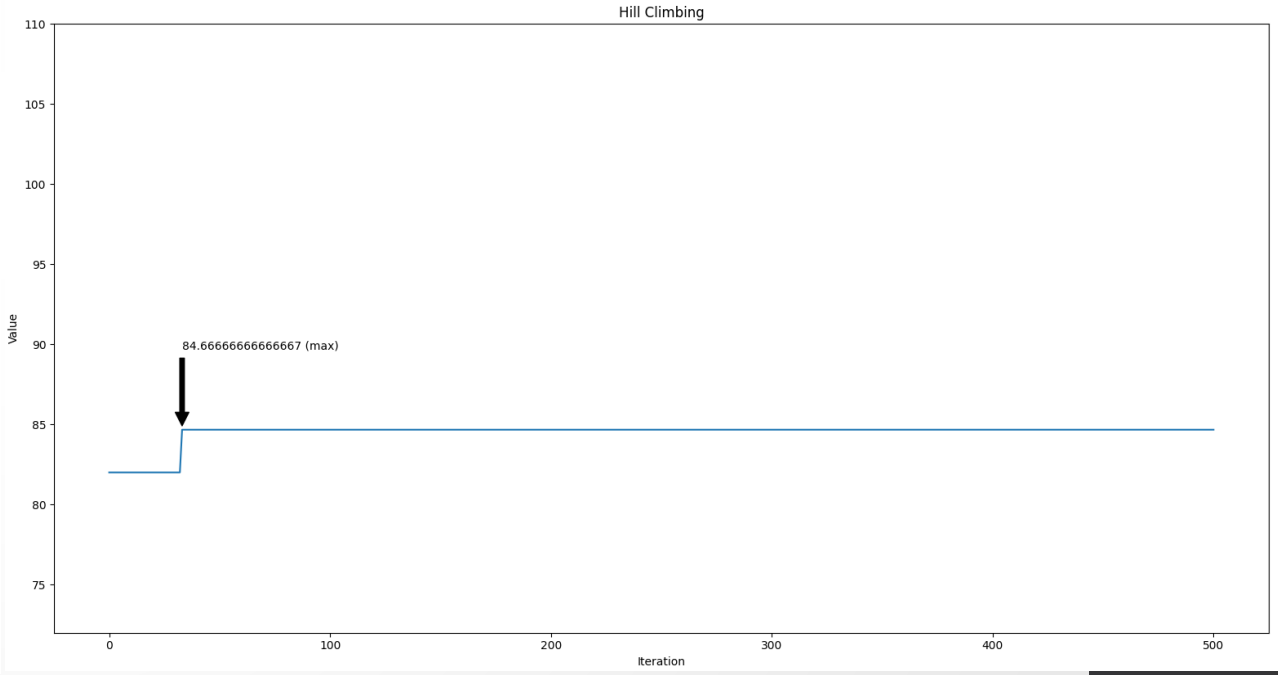


Fig7 – **Hill climbing algorithm graphic**, starting with a solution with score 83, and ending with 84.66

Experimental Results

File: busy_day (30 drones, 400 product types, 10 warehouses, 1250 orders)

Iterations: 100 Simulated Annealing iterations – with 250 iterations each one

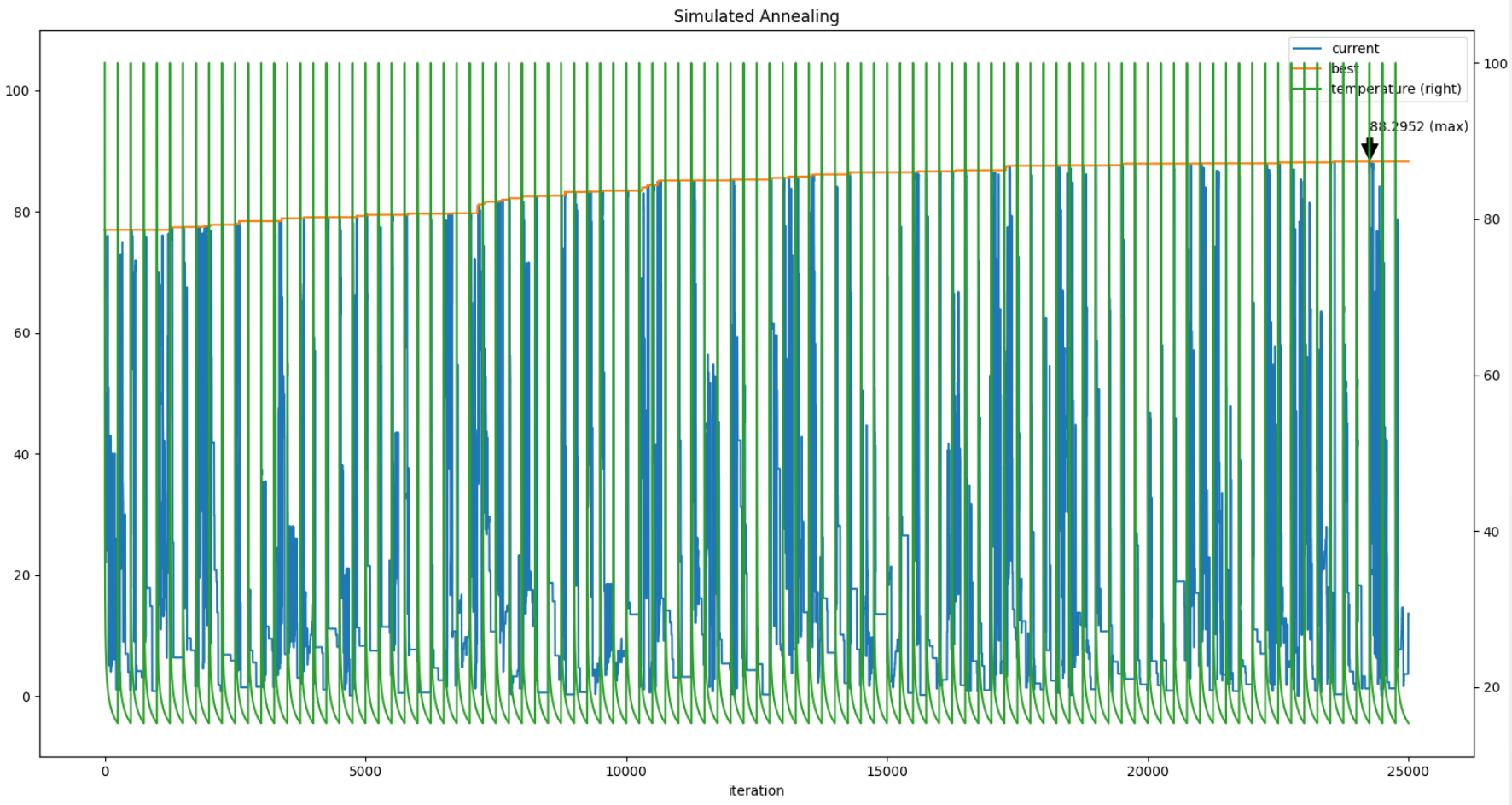
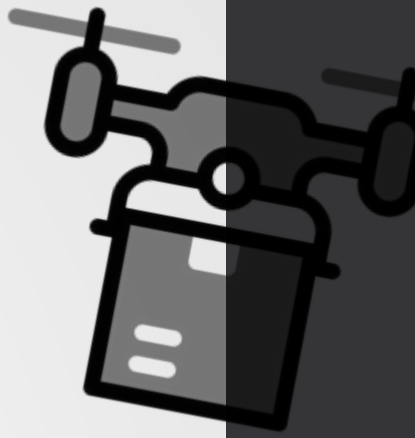


Fig8 – **Simulated Annealing algorithm graphic**, starting with a greedy solution with score 76, and ending with 88.29

Conclusions

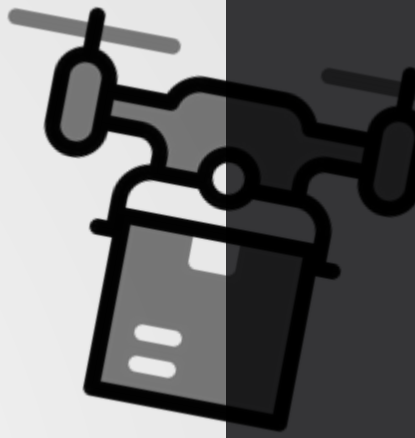


In conclusion, after testing the implemented algorithms with all the input files we noticed that the Iterative Simulated Annealing algorithm is the one that most improves the initial solution, obtained with the greedy or naive algorithms.

Furthermore, the mutations and crossovers on the Genetic and mutations on the Simulated Annealing algorithms were causing them to accept Chromosomes with a very low score, therefore, a lower bound limit was defined on these two algorithms to prevent them from working with impracticable solutions.

In our problem’s case, it is better to have a costly algorithm which finds one solution, and have that be a good solution, than to have a meta-heuristic try to enhance an average solution.

Related Work and Materials



2020's Kaggle approach of same problem, created by Google: https://www.kaggle.com/c/hashcode-drone-delivery

First Place solution, using Scala: https://www.kaggle.com/lyxthe/1st-place-solution | https://github.com/miraan/google-hash-code-qualification-round

Second Place Solution, using Python: https://www.kaggle.com/royceda/drone-linear-prog-and-routing-heuristic-90-done

Genetic Algorithm

A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries (2011): https://www.sciencedirect.com/science/article/pii/S036083521100348 2

A vehicle routing problem solved by using a hybrid genetic algorithm (2007): https://www.sciencedirect.com/science/article/pii/S0360835207001222

A hybrid genetic algorithm for the multi-depot vehicle routing problem (2005): https://www.sciencedirect.com/science/article/pii/S0952197607000887

- **Programming Language:** Python, with plot visualization using **MatPlotLib** library.
- **Development Environment:** JetBrains IntelliJ