U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Drone Delivery

Metaheuristics for Optimization Problems

Artificial Intelligence

Integrated Masters in Informatics and Computer Engineering

Ana Teresa Cruz - up201806460
André Nascimento - up201806461
António Bezerra - up201806854
Group 25

Specification

Given a fleet of drones, all with a determined finite capacity, a list of customer orders and availability of the individual products in warehouses, schedule the drone operations so that the orders are completed as soon as possible.

All drones start from the same warehouse and will pick up items from that warehouse that will be delivered to customers or other warehouses.

Not all orders have to be fulfilled.

The problem can be interpreted as a **Vehicle Routing Problem (VRP)**.

Problem Formulation

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Evaluation function: For an order completed in turn *t* and a simulation taking *T* turns in total, the score for the order is calculated as $(T - t) / T \times 100$, rounded up to the next integer. The score of a simulation is the sum of the score of all orders. An unfulfilled order has score 0.

Rigid constraints:

- Drones cannot carry more than their capacity;
- Each drone's operations cannot exceed the maximum number of turns;
- Drones can only load products in Warehouses;
- Drones cannot load more products than are available at a Warehouse;
- Drones can only deliver or unload products that they are carrying.

Problem Formulation

Solution representation:

Drone	1		2			3
Warehouse	W1		W2			W2
Order	O1		O2			O1
Product	1	2	2	3	4	2
Quantity	1	1	2	1	1	1

Shipment

...

Neighborhood/mutation and crossover functions:

Mutations:

- Change the drone that performs a shipment;
- Swap a shipment between two drones;

Crossover:

- Two point crossover.

Note: This representation only considers paths of the type W -> O and, for each Shipment, all the products loaded into the drone at the Warehouse are delivered at the Order. This restriction may decrease the quality of the solutions but it greatly reduces the processing time and the possibility of generating invalid solutions, since it makes it simpler to enforce the constraints.

Work Already Done

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Programming language: **Python**

IDE: **IntelliJ/VSCode**

Data structures:

- Python classes for the various problem entities
- Lists for storing the problem domain and the solution (list of drone commands)
- Graph that represents the map

File structure: specified in the HashCode problem statement.

Implemented:

- File parsing

Developed Approach

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Heuristics:

- *Greedy heuristic*: attributes shipments to drones to fulfill available orders, choosing the immediately most advantageous pairing of warehouse and order for each drone.
- *Random heuristic*: attributes random shipments to each drone, used to provide an initial population for genetic algorithms.

Evaluation functions: Shipments are ranked based on a partial score calculated using the percentage of the order they fulfill (in weight) and the number of turns needed to perform it. The final solution is evaluated using the function described previously.

Shipment construction:

- *Largest first*: orders products by weight.
- *Knapsack*: maximizes occupied capacity.

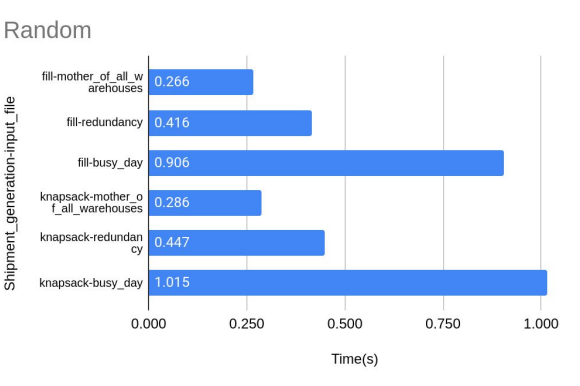
Implemented Algorithms

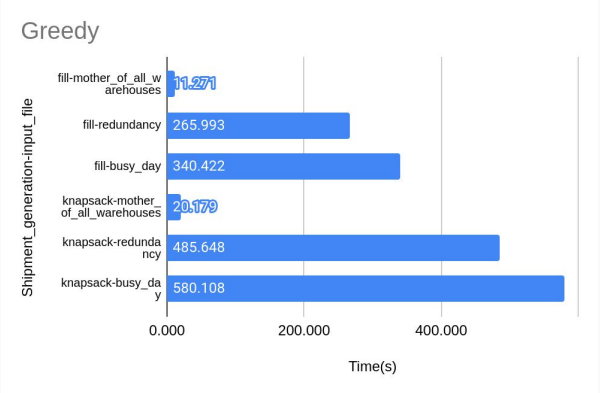
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

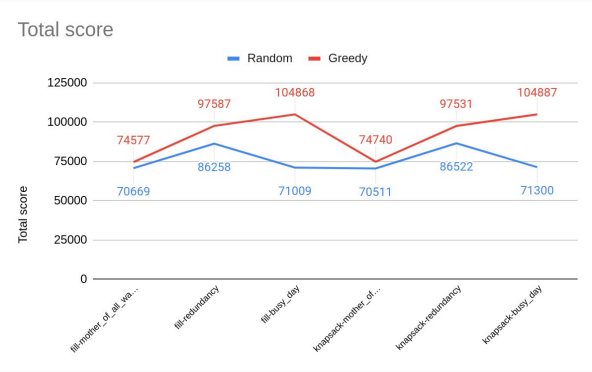
Metaheuristic	Description	Parameters	Code
Hill-climb Random	Find a random neighbor of the current solution using a mutation function. Evaluate and keep it if is better than the current one.	Number of iterations, probability of chosen mutation.	github
Simulated Annealing	Find a random neighbor of the current solution using a mutation function. Evaluate and keep it if is better than the current one or, if it is worse, based on a probability inversely proportional to the *temperature*.	Starting temperature, cooling schedule, number of iterations, minimum temperature, number of runs, probability of chosen mutation.	github
Genetic Algorithm	Generate an initial population randomly or seeding with some previously generated solution. Evolve the population by performing crossover and mutations on certain individuals. Of the new generation, select the surviving chromosomes based on a roulette, with the probability proportional to each chromosome's score.	Initial population, size of population, number of generations, size of crossover, probability of crossover, probability of mutation, probability of chosen mutation.	github

Experimental Results

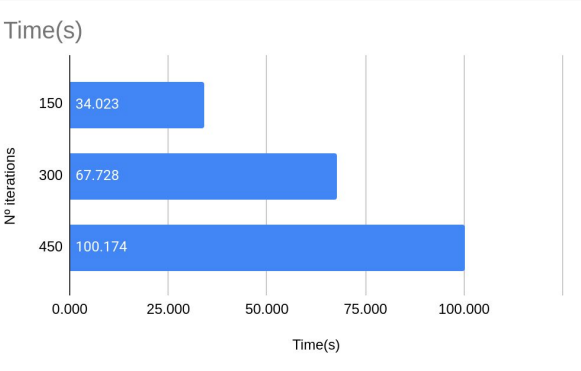
Heuristics

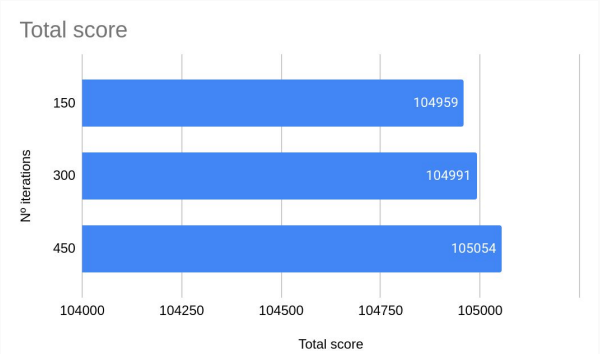






Hill Climbing





Full results:
https://docs.google.com/spreadsheets/d/19mv-KiGjx-yVYyh83nGSNyaaGsYPDwQfGgteP_Q1Hls/edit?usp=sharing

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Simulated Annealing (busy_day, maximum of 1000 iterations)			
Cooling function	**Min temp**	**Time (s)**	**Total score**
exponential	0.01	9.56	104887
	0.005	10.232	104887
	0.001	11.652	104904
logarithmic	0.01	232.693	104887
	0.005	232.151	104887
	0.001	225.65	104887
linear	0.01	228.243	105154
	0.005	222.412	105191
	0.001	234.903	105076
quadratic	0.01	23.075	104943
	0.005	31.727	104935
	0.001	71.53	105093

Simulated Annealing (busy_day, quadratic, min-temp 0.001)		
Nº iterations (recursive)	**Time (s)**	**Total score (final)**
2	151.7	105111
3	227.405	105181
4	295.121	105145

Genetic (busy_day)						
Population Size	**Nº Iterations**	**Prob. Crossover (%)**	**Prob. Mutation (%)**	**Penalty**	**Time (s)**	**Total score**
25	30	5	20	0	89.844	71112
50					175.789	71501
100					334.453	71465
50	15	5	20		126.855	71648
	30				172.785	71091
	50				237.087	70274
50	30	5	20		176.97	71934
		10			221.084	71276
		20			282.124	68742
50	30	5	5		111.976	72165
			10		138.745	71336
			20		168.661	72172
50	31	5	5	-10	171.062	71587
				-25	171.073	71301
				-50	167.97	71113

Conclusions

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- We developed effective heuristics and metaheuristics to solve the problem and implemented different metaheuristics.
- Our simplified representation made the problem much easier to solve from the perspective of ensuring constraints were respected.
- While a strong enforcement of constraints avoided generating solutions that were very weak because of excessive invalidity, this also limited the quality of our solutions.
- The most evident case was the Genetic Algorithm, where the constraints limited our ability to generate better mutations or obtain more efficient paths through crossover.
- In a future work, implementing a different representation, or extending the current one to allow for more flexibility, might be useful for generating better results.
- The experimental results show reasonable performance in terms of time complexity. In the case of the genetic algorithm, this was in great part due to the use of concurrency for generating solutions.
- This project allowed us to deepen our knowledge of the implemented algorithms and also gave us a real-world example of their shortcomings and specificities.

References

- Google HashCode problem statement:
  https://storage.googleapis.com/coding-competitions.appspot.com/HC/2016/hashcode2016_qualification_task.pdf
- Kaggle competition:
  https://www.kaggle.com/c/hashcode-drone-delivery/code?competitionId=22040
- Articles about the CVRPPAD:
  https://link.springer.com/content/pdf/10.1007/s10479-017-2722-x.pdf
  https://www.sciencedirect.com/science/article/pii/S0957417420307429
  https://www.wt.pw.edu.pl/index.php/layout/set/print/content/download/2792/17069/file/Z97-art_18.pdf
  http://www.riejournal.com/article_47997_08a232e0e64857a009a93b3355e4cd54.pdf
- OR-Tools Knapsack Solver:
  https://developers.google.com/optimization/bin/knapsack
- Numpy random library:
  https://numpy.org/doc/stable/