AEA – Homework 2

The problem to solve was TSPD. As this problem is known to be NP-hard, a deterministic approach for this problem would give either a far from the optimum solution, either a way too much time consuming solution. For this reason, we have implemented 2 heuristic solutions to solve our problem: a Genetic Algorithm (GA) and an Ant Colony Optimization (ACO or simply AC). Another solution implemented is the use of Constraints Programming, but the solutions it gave us were provided too slowly, so we didn't compare the GA and the ACO with it.

Both the Genetic Algorithms and Ant Colony Optimizations are nature-inspired metaheuristics, and both of them swarm around the notion of individual (in ACO it is also called an ant). This individual is evaluated based on its structure, and, afterwards, various search methods are used. To search the solution, one must define the structure of the individual; this structure is common for both the GA and the ACO:

Structure

- $individual = (permutation,\ drone\_visits) = (p, v)$
- The number of cities is n
- Permutation $= permutation = p$:
  - a permutation of the list $[1, \dots, n\ -\ 1]$, appending a 0 at the beginning and the ending of the permutation => $p = [0, i_1, i_2, \dots, i_{n-1}, 0]$, where $i_j \in \{1, \dots, n-1\},\ j \neq k \Rightarrow i_j \neq i_k,\ \forall j, k = \overline{1, n-1}$
- Nodes visited by drone $= drone\_visits = v$:
  - A list of Booleans. For each $i = \overline{1, n-1}$, we have $v_i \in \{0, 1\}$, and we have $v_0 = v_n = 0$ => $v = [0, v_1, v_2, \dots v_{n-1}, 0]$, where $v_i \in \{0,1\}$
  - The value of each element v_i from this list creates 4 cases, where $i = \overline{1, n-1}$:
    - The drone is in the truck and $v_i\ =\ 0$: The drone does nothing while the truck goes from p_{i-1} to p_i.
    - The drone is in the truck and $v_i\ =\ 1$: The drone goes from p_{i-1} to p_i, while the truck does nothing
    - The drone is not in the truck and $v_i\ =\ 0$: The last action of the drone was deliver a parcel; the drone does nothing, while the truck goes from p_{i-1} to p_i
    - The drone is not in the truck and $v_i\ =\ 1$: The last action of the drone was to deliver a parcel; the drone and the truck are at different houses; they both go, from their current place, to p_i (exactly one of them will be at p_{i-1} before going to p_i)
  - this list starts and begins with 0

- basically, if there's an even number of 1's in $[v_1, \dots, v_i]$ both the drone and the truck are at the same house i; if there's an odd number of 1's, it means the drone and the truck are at different houses
- if the number of values 1 in the whole list is odd, the truck and the drone meet at node 0 (basically, if the numbers of 1 is odd, the last value of the list becomes 1 instead so the number of 1's will always be even = the truck and the drone will always sync)

This structure gives us the possibility to create and compute any solution for our problem TSPD. These give us the following:

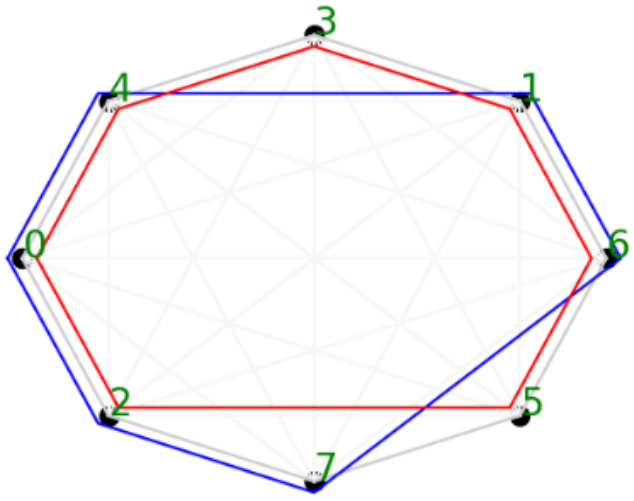
- Truck path:
  - the path the truck takes
- Drone path:
  - the path the drone takes

- The total time will be

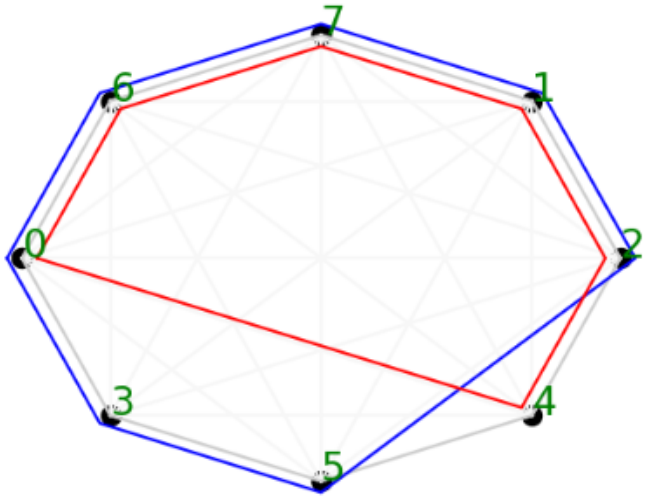
$$\max(length(truck\_path), length(drone\_path))$$

To better view how all of these are defined, observe the 2 following examples in the pictures below:

```
[0 4 3 1 6 5 7 2 0] Permutation (city visiting order)
[0 0 1 1 0 1 0 1 0] Nodes visited_by_drone
[0 4 1 6 7 2 0]     Truck Path (blue line)
[0 4 3 1 6 5 2 0]   Drone Path (red  line)
```



```
[0 6 7 1 2 4 5 3 0] Permutation (city visiting order)
[0 0 0 0 0 1 0 0 0] Nodes visited_by_drone
[0 6 7 1 2 5 3 0]   Truck Path (blue line)
[0 6 7 1 2 4 0]     Drone Path (red  line)
```



Genetic Algorithm

For the genetic algorithm, we need two separate sets of operations: one set for the permutation part, one set for the drone visiting part. Parameters used:

POP_SIZE = 200

N_GENERATIONS = int(3000 / ln(n))

PROBABILITY_FOR_INDIVIDUAL_TO_BE_SELECTED_FOR_MUTATION=5%

PROBABILITY_FOR_ELEMENT_OF_INDIVIDUAL_PERMUTATION_TO_BE_MUTATED=1%

PROBABILITY_FOR_ELEMENT_OF_INDIVIDUAL_DRONE_VISIT_TO_BE_MUTATED=5%

PROBABILITY_FOR_CROSSOVER=20%

Mutation

Permutation: the mutation is done by switching 2 random values=city numbers from the permutation (Shuffle Indexes mutation).

Drone visits: the mutation is done by considering the values 1 and 0 as True and False, and flipping the value of this.

Crossover

Permutation: the crossover, known as the Partially Matched Crossover (PMX), is widely used in permutation based individuals. It is well explained at the following location: https://www.researchgate.net/figure/Partially-mapped-crossover-operator-PMX_fig1_226665831
Drone visits: the crossover, known as Two Point Cut crossover, switches from the randomly set cut point 1 to the randomly set cut point 2 the content of the 2 individuals, only between those 2 cut points.

Selection

The selection works on the whole individual, so it is not separated for the permutation and drone visits parts. Tournament selection was used, with tournament size = 3.

Evaluation function

This is the most important part of the genetic algorithm. It is computed as specified above, in the **Structure** chapter.

Ant Colony Optimization

For the ACO solution, we have implemented some small variations to it. The whole algorithm is briefly described below:

- Parameters used:
  POP_SIZE=10
  N_GENERATIONS= int**(300/** ln**(n))**
  PHEROMONE_DECAY=0.1
  CONSTANT_HEUR=2.5
  CONSTANT_LOCAL_PHEROMONE=0.1
  CONSTANT_GREED_CHANCE=0.7

- The ants are initialized with the structure described above
- An initial best solution is chosen from the first ants, and based on its structure cost, an initial pheromone level is set on the pheromone matrix
- After the initialization, for each generation, each ant heuristically tries to find the best route. For this, at each step it takes into account the pheromone levels matrix and the best option from all the options to choose from. With these computed, with a probability equal to **CONSTANT_GREED_CHANCE** it chooses that best option mentioned beforehand, and with a probability **1 – CONSTANT_GREED_CHANCE** it chooses one random option

- After each candidate solution found, the pheromone matrix is updated based on that solution's route cost (known as local update)
- After each iteration, the pheromone matrix is updated based on the best solution found so far (known as global update)

Implementation

Both of these algorithms were implemented in Python 3.6.3 (Anaconda distribution); although no Python 3.6.3 specific syntax or libraries were used, which means it can be run on many other Python versions. The implementation also uses the Python libraries numpy==1.17.0 and deap==1.2.2 (numpy for fast n-dimensional array computations and deap for the Genetic Algorithm operations).

Computational Experiments

The implementations were executed on a Dell E7470 laptop (Intel i7-6600U CPU – 2.60GHz - 4CPUs, 16GB RAM). Both have been run 10 times on 5 different randomly generated datasets, with distances between the cities taking integer values between 1 and 99. The datasets are:

- Very small: 9 nodes
- Small: 17 nodes
- Medium: 42 nodes
- Large: 60 nodes
- Very large: 563 nodes

Statistical results can be found in the table below (the time is specified in seconds):

Algorithm	Data set	Average cost	Average time		Minimum cost	Minimum time		Maximum cost	Maximum time		STDDEV cost	STDDEV time
ACO	Very small	170.7	1.12		164	1.06		177	1.2		5.22	0.04
ACO	Small	197.2	3.53		178	3.17		210	4.5		10.07	0.42
ACO	Medium	298	16.29		277	15.55		318	19.26		13.52	1.02
ACO	Large	407.6	28.87		376	27.65		438	29.88		17.02	0.67
ACO	Very large	2098.7	2178.8		1865	2158.49		2569	2258.03		213.07	27.63
GA	Very small	196	17.04		164	12.37		221	24.31		14.78	3.57
GA	Small	246.7	13.64		222	12.23		267	17.56		13.48	1.6
GA	Medium	733.8	17.78		667	16.61		831	19.46		50.21	0.81
GA	Large	1284.3	21.21		1122	19.65		1430	22.57		101.52	0.74
GA	Very large	22035.7	101.98		21144	98.08		22566	106.5		461.14	2.51

As it can be seen, the time it takes to run varies greatly, based on the city matrix size, for the ACO implementation, while for the GA implementation not. This is because the ACO searches, at each step, for the best next step (the greedy part) => quadratic complexity, while the GA executes its steps indefinitely, not searching for anything => linear complexity.

Looking at the evaluation functions (cost) for the solutions found, it can be seen that the ACO implementation finds way better solutions than the GA, especially as the size of the city matrix grows. This is because the GA basically generates random solutions and tries to improve them little by little, not necessarily well suited for permutation based cost functions; in the meantime, the ACO searches greedily for a good solution, as it is taking into account past good solutions (based on the pheromone matrix) and also possible new better solutions.

The random search done by the GA can also be seen in the Standard Deviation cost column, where the costs of the solutions found have a greater standard deviation than the costs of the solutions found by the ACO. On the other hand, observing the Standard Deviation time column, it can be seen that any two executions of the GA take almost exactly the same time, compared to the ACO, which highly depends on the previous candidate solutions found (the best solution is the same for many iterations, and this solution's cost is only computed once). (The GA executions for the very small and small data sets were the first to be run, so the time was influenced by some other operations on the machine at that time => for these 2 the standard deviation time is very high).

Conclusion

Both algorithms implemented to solve TSPD are viable, robust and easy to use. They only need the number of cities and the distance matrix as input.

If working with small distance matrix (say, $n < 50$) ACO is better than GA. For $n > 50$ we should take into account the spare time we have: ACO gives way better solutions than the GA (10 times smaller routes than the GA gives for the very large data set), but take way more time to run (20 times more time taken for the ACO to run for the very large data set).