Name: Mai Tung Duong
ID: 20180745
Report: Stochastic Optimisation

A- Code documentation

1. Structure and how to run:
   The solver implements 3 algorithm: Genetic Algorithm, Particle Swarm Optimization and Ant Colony Optimization
   To run the code:
   Step 1: Open the terminal for tsp/ directory
   Step 2: Put the .tsp file in the same directory
   (I include burma14.tsp, berlin52.tsp, att48.tsp, ch130.tsp and a280.tsp if you want to test immediately. For other files, please put it yourself)
   Step 3: Install dependencies
   pip install -r requirement.txt
   Step 4: Run
   Refer to the runner interface.
   Example: The command to run burma14.tsp with default parameters is
   python solver.py burma14.tsp
2. Dependencies: NumPy, matplotlib, scikit-learn
   pip install -r requirement.txt
3. Runner interface:
   python solver.py source [-p P] [-f F] [-solver SOLVER] [-graph GRAPH]
   [-prim PRIM] [-pressure PRESSURE] [-w W] [-c1 C1] [-c2 C2]
   [-a A] [-b B] [-r R]

   Positional arguments: source Input file

   Optional arguments:

Arg	**Meaning**	**Default**	**Type**
-p P	POPULATION	10	int
-f F	MAX FITNESS CALL	100000	int
-solver SOLVER	solver (algorithm to use)	ACO	str
-graph GRAPH	Draw graph or not	False	bool
-prim PRIM	Initialize using Prim algorithm or not	False	bool
-pressure PRESSURE	GA selection pressure	0.5	float
-w W -c1 C1 -c2 C2	PSO weights	4 2 1	float
-a A -b B -r R	ACO weights (alpha, beta, rho)	1 3 0.1	float
-v V	Verbosity	False	bool

   Example: To solve burma14.tsp with GA, show final graph and all other default settings
   python solver.py burma14.tsp -solver GA -graph True

B- Implementation

Main approach: Utilize bio-inspired algorithm

I- Loss/fitness function:
   Loss/fitness function is the tour length. We want to MINIMIZE this value (more like a loss)

II- Solution representation

The Solution class has 2 important attributes:

- Path: The sequence of cities in the tour, starts and ends with 0 (the first city numbered with 0). When exporting the result, the city will be add 1 (so the first city is 1) to match with the original numbering of .tsp file
- Score: Value of loss/fitness function of the path

III- Population representation and initialization

Population is simply a list of Solution instances

I implement 2 way to initialize the population:

- Stochastic: Just make several random paths (albeit still try to make the probability inversely proportional with the distance)
- Prim-inspired: (mixture of stochastic and deterministic):
  Half of the population are initialized stochastically
  Find a 2-approximation of the problem by the famous Prim MST algorithm
  The other half of the population is essentially this solution with some additional random noise (by mutation or Gaussian noise …)

IV- Optimization algorithm

1. Genetic Algorithm:

a. Algorithm: Similar with algorithm that professor use to solve the password

Gene operator:

- Selection: tournament selection
  We choose a set of group_size solution instances, and choose the best one among them
- Crossover:
  Randomly choose the position to flip the node between two parents
  Leave the node empty if the node in the other parent already in the path
  Finally, we fill those empty nodes with remaining cities.
- Mutation:
  Simply exchange 2 cities in the path

In each iteration:

- Create new children until the population double:
  - We pick 2 solutions by tournament selection to be parents
  - Crossover to create two children
  - Mutate two children
- Discard half of the current population to maintain the population size, with some selection pressure. This pressure determines the ratio that we choose good solutions greedily. The rest will be chosen randomly from the remaining solutions.

b. Rationale: We keep good solution in the population yet still keep some seemingly bad solution but might be beneficial in later crossover and mutation (maintain diversity)

c. Complexity: For each iteration and each solution, the complexity is O(n) as the calculation is done on the path of size n+1 (mutation O(n), selection O(1), crossover O(n))

2. Ant Colony Optimization:

a. Algorithm: Exactly the same algorithm introduced in the lecture.

Each Solution is considered as the trajectory of an ant.

For an edge, the visibility η is defined as inverse of its length.

Initially, we add a unit of pheromone to every edge and calculate the weights as:

$$w[i][j] = \frac{\tau[i][j]^{\alpha}\eta[i][j]^{\beta}}{\sum_{*}\tau[i][*]^{\alpha}\eta[i][*]^{\beta}}$$

Here, τ is the amount pf pheromone, α, β are control hyperparameters

The weight of every edge is then represented as a symmetric adjacency matrix (w[i][j] denotes the edge that connects i and j).

In each iteration, for every ant:

- We drop the ant at city 0 and let it move stochastically with probability proportional with the weights of edges (only consider those edges leading to unvisited cities). The last city will be connected back to city 0 to make a tour
- We add the amount Q/L of pheromone where Q is the current shortest tour and L is the tour length
- Then, some of the pheromone evaporates with rate ρ: $\tau := (1-\rho)\tau$

b. Rationale: Ideally, the good tour will be found as the pheromone will be added quickly for the short path, making a heavy weight on those edges.

c. Complexity: For each iteration and each solution, the complexity is $O(n^2)$ as the calculation is done on the weight matrix of size $n \times n$

3. Particle Swarm Optimization

a. Algorithm: The same algorithm introduced in the lecture.

First, we define **position**: Position is defined as a weighted adjacency matrix (weights[i][j] denote the weight of edge connecting i and j)

Velocity is the change in this position.

We encode the position by greedily follow the edge with maximal weight (among edges leading to unvisited cities) to get the path.

Assign w, c_1, c_2 as the weight, r_1, r_2 as random numbers, pbest as local best, gbest as global best

For each iteration and each solution, velocity and position is updated:

$$v_i^{t+1} = wv_i^t + c_1 r_1\left(pbest - x_i^t\right) + c_2 r_2\left(gbest - x_i^t\right)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

To ensure the non-negative and bound value of position, a RELU + normalization is applied on the new position.

b. Rationale: Ideally, the good tour will be found as the velocity will push them closer to global and local optimum, but the randomness added could still 'overfly' the local optimum and hopefully find the global optimum.

c. Complexity: For each iteration and each solution, the complexity is $O(n^2)$ as the calculation is done on the weight matrix of size $n \times n$

C- Some demo result:

Note:

- Optimum tour for burma14, berlin52 and ch130 has length 30.88, 7542 and 6110.
- All parameters are at default values if not specified.

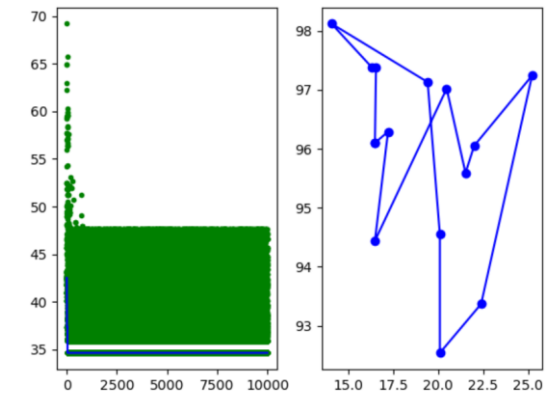
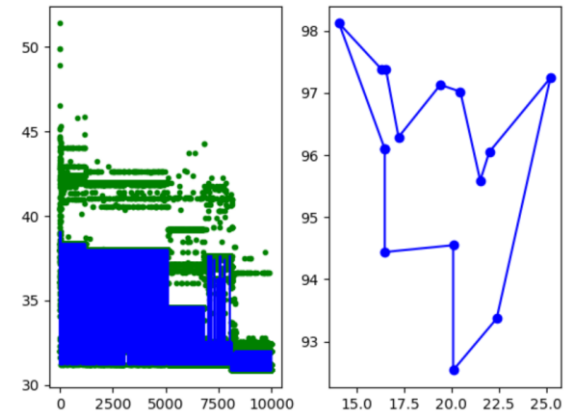
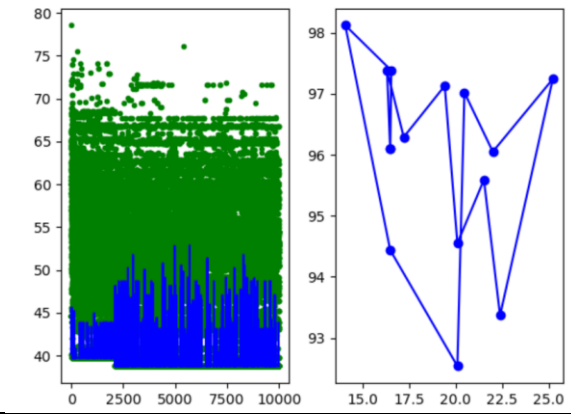
Stochastic

	ACO		GA		PSO	
burma14	30.88	46.8s	36.34	23.2s	36.33	31.0s
berlin52	7831.0	267.5s	9402.2	44.0s	19146.7	147.5s
ch130	6645.5	1259.0s	10658.0	156.7s	37972.4	693.6s

Semi stochastic (Prim-inspired)

	ACO		GA		PSO	
burma14	31.22	39.8s	32.25	25.4s	40.2s	68.1s
berlin52	7771.0	89.0s	8595.5	59.4s	12082.4	170.8s

Graph for burma14, stochastic:
Left graph: Green are solutions, blue is the best solution in the population,
Right graph: The path in 2-D Euclide space

GA	
ACO	
PSO	

Via these results, it seems like ACO performs the best but slow, GA perform reasonably well and fast. PCO does not work well.

ACO performs well even without Prim initialization. Initialization seems to save time for ACO and improve the result for GA

Looking at the graph it is quite clear that ACO has the best evolution: good diversity as well as good optimization