Excercise 4
Implementing a centralized agent

Group №89: 309398, 270000

November 3, 2020

1 Solution Representation

1.1 Variables

Our solution representation makes us of the following variables while the `TaskAnnotated` class is a class that regroups a `logist.Task` with an action a to perform on it, either pick up or deliver. Also note that the `vehicle` is a redundant variable which can be deduced using other two variables.

- `nextTaskTask`, a HashMap <`TaskAnnotated, TaskAnnotated`> of $2 * N_T$ keys, that maps a given task action to its next one.
  $nextTaskTask(t) \in \{t_1(p), t_1(d), ..., t_{N_T}(p), t_{N_T}(d)\}$
- `nextTaskVehicle`, a HashMap < `Vehicle, TaskAnnotated`> of N_V keys, that maps a vehicle with its first task action.
  $nextTaskVehicle(v) \in \{t1(p), t1(d), ..., t_{N_T}(p), t_{N_T}(d)\}$
- `vehicle`, a HashMap < `TaskAnnotated, Vehicle` > that maps a task action with its vehicle. $Vehicle(t) \in \{v_1, ..., v_{N_V}\}$

1.2 Constraints

- For each task t_i, $t_i(p)$ must be before $t_i(d)$ and need to be performed by the same vehicle.
- $nextTaskVehicle(v_k) = t(p)$: the first task action of each vehicle needs to be a pickup action.
- $nextTaskTask(t_i(a)) \neq t_i(a)$, the action performed after a given action cannot be the same.
- $nextTaskTask(v_k) = t_j(a) \Rightarrow vehicle(t_j) = v_k$
- $nextTaskTask(t_i(a)) = t_j(a) \Rightarrow vehicle(t_j) = vehicle(t_i)$
- all the tasks need to be delivered; the set of values of $nextTaskTask$ and $nextTaskVehicle$ combined must be equal to the set of T tasks with both actions plus N_V times the value $null$.
- the capacity of a vehicle cannot be exceeded.

1.3 Objective function

$$C = \sum_{i=1}^{N_V} dist(home(v_i), nextTask(v_i)_{pickupCity}) * cost(v_i) + \sum_{i=1}^{2N_T} dist(t_j, nextTask(t_j)) * cost(vehicle(t_j))$$

where $cost(v_i)$ is the cost per kilometer of v_i, $dist(t_j, nextTaskTask(t_j)$ is the distance between the pickup or delivery city of t_j and the pickup or delivery city of $nextTaskTask(t_j)$ depending on both task actions. It is important to know that we are not interested in the speed of the delivery, hence the vehicle speed is not used for the objective function calculation.

2 Stochastic optimization

2.1 Initial solution

We first assign vehicles pickup tasks to tasks whose pickup city is the vehicle's home city given the capacity is available. Once finished we then add the delivery tasks for the tasks already assigned to vehicles making sure that the vehicles capacity will be fully available for next tasks. Then we randomly distribute the remaining tasks among vehicles with pickup task and delivery task for each task given the vehicle capacity is enough to fit the task. The cost of our initial solution revolves around 44711.

2.2 Generating neighbours

We generate neighbours by applying the following transformations:

- Change the task action order in a vehicle : we choose a vehicle at random and change its plan order. We generate every possible combination of pairs of task that can be exchanged. The simple logic behind exchange is it is okay to postpone a delivery or do an early pickup, given the capacity is available. However if we are planning to do an early delivery or a late pickup we should make sure that for each task the pickup is always happening before the delivery.

- Move a task action from a vehicle to another: we choose a vehicle v_i at random that has a plan. Then for every possible task t_j the vehicle takes care of, we move it from v_i to every other vehicle as its first task as long as the receiving vehicle's capacity allows for it.

2.3 Stochastic optimization algorithm

After computing the initial solution, the algorithm explores the solution space by applying local transformations to a candidate solution. From the computed set of neighbours, it moves to a solution with the minimal cost and repeats the process by generating new neighbours. If we find other neighbours with same minimal cost, we randomly pick one among the best ones. To avoid falling into a local minima, we discard the best neighbour with a probability $1 - p$ and stick with the previous solution. However, we always keep the best solution found so far while going through this loop and when the algorithm finishes, we return this optimal solution.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

The experiment is performed on the England's topology with 30 tasks with the same weight of 3 and 4 vehicles with the same load capacity of 30. We experiment with different probabilities p of moving towards the best solution. We run the algorithm for 1000 iterations for comparison (in-expense of the best solution found) even though in default case it stops at the default timeout plan of 300 seconds.

3.1.2 Observations

We obtain the best mean optimal cost of 20398 with $p = 0.7$ as there is a right balance between exploring the solution space and improving on the solution. Additionally, we obtain costs of 22737, 24618, 22881 and 25795, for $p = 0.1, 0.3, 0.5, 0.9$ respectively. $p = 1$ leads to the worst performance of 26292.5 because always selecting the best neighbour leads the algorithm to be stuck in a local minima.

3.2 Experiment 2: Different Vehicle configurations

3.2.1 Setting

In this experiment we are using the same setting as 3.1, but with different values for each vehicle's capacity, speed and cost-per-km and number of vehicles. In the first run we will use capacities 2, 5, 10, 20, 30, 40, 50 and 100 for all the vehicles for each run and one run with four vehicles having 5, 20, 40 and 55 (average 30) capacities. Then we will do the experiment with different cost-per-km 1, 3, 7 and 9 (average 5) for each four vehicles respectively. Then we will do the experiment with total 1, 2, 4, 8 vehicles at the map, each origin at different cities.

3.2.2 Observations

For different capacities we got an termination for 2 and a cost of 50460, 41729, 25339, 22881, 22840, 23810, 22250 for rest. This is because there is no solution if capacity is smaller than task and then because once a vehicle have enough capacity, there is no added advantage being vehicles have extra capacity while having a similar total weight for all the tasks. For different capacities we got a cost of 25571 which is a bit higher for the average 30 capacity case and can also see first two vehicles stops early giving bigger vehicles to take more tasks. With different cost-per-km cost was 12321, which was significant deduction and possible because now it will mainly use first two vehicles to deliver tasks. For different vehicle count, cost was 27133, 23216, 22881 and 21851 which was a monotonically decreasing function. However even in the 8 vehicle case there were two vehicles not moving.

3.3 Experiment 3: Different Task configurations

3.3.1 Setting

In this experiment we are using the same setting as 3.1, but with different configuration for task distribution such as number of tasks and different distributions for each task's weight. First we will do an experiment with constant weight of 3 per task for number of tasks 5, 10, 20, 30, 40, 50 and 100. Then we will do the experiment with a fixed number of tasks of 30, but with a uniform distribution of task weights between 1 and 5.

3.3.2 Observations

The cost for number of tasks 5, 10, 20, 30, 40, 50 and 100 was respectively 5795, 10615, 16250, 22881, 30071, 37220 and 108847 which gives 1159, 1061.50, 812.50, 762.70, 751.78, 744.40, 1088.47 cost per task. We can see a reduction in cost per task when number of task increases because this better fit our initial solution with picking up task at origin-city first and divide rest randomly. However when the task count is even higher, there will be more tasks at origin city forcing more random distribution of tasks.

Then with the random weight distribution of task, we got a cost of 26171 which is a bit higher than the cost for fixed weight of 3 which is also the mean weight of this run. This is expected as we did not account for weight distribution on our initial solution. However since it is captured in the objective function, the ultimate solution should be optimal for the random weights of tasks.