Excercise 4
Implementing a centralized agent

Group №9: Fengyu Cai, Wanhao Zhou

November 5, 2019

1 Solution Representation

1.1 Variables

We first define the basic action of a vehicle irrespective of the movement between cities, considering only `Pickup` and `Delivery`. We call this basic action `CentralizedAction`, consisting of two properties: `task` and `isPickup`, i.e. for a given task, we consider two actions and thus this enables our vehicles to carry multiple tasks, as long as it does not reach the maximum capacity, without the need to deliver first.

For each vehicle, we assign a list of `CentralizedAction` to demonstrate its plan. Together, all vehicles' plans are represented by a `HashMap`, the key being the vehicle. We define a `CentralizedState` containing such hash map and the cost of the current state. In our algorithm, we need to maintain the feasibility of the state by satisfying constraints and try to minimise the cost of it.

1.2 Constraints

We need to satisfy the following constraints to maintain the feasibility of the solution:

- All tasks are assigned to some vehicle.
- For the actions of a certain vehicle, the `Pickup` must be carried out before `Delivery` on any task.
- At any time and for any vehicle, the load of tasks must not exceed its capacity.

1.3 Objective function

Denote the vehicle set as V. $A(v)$ represents the number of actions of vehicle v. $\text{City}(\cdot)$ is a function that gets the respective `Pickup` or `Delivery` city of a task depending on the type of the action. Consider only vehicles having a non-empty plan, we define the objective function J to minimise as follows:

$$J = \sum_{v\in V} \text{Cost}(v) \sum_{i=0}^{A(v)-1} \text{Dist}\big(\text{City}(i),\ \text{City}(i+1)\big)$$

2 Stochastic optimization

2.1 Initial solution

For any given task, we randomly choose a vehicle to pick up and deliver the task. Notably, we assign two consecutive actions `Pickup` and `Delivery` at the same time so that we do not need to check the capacity constraint. At this time, each vehicle delivers tasks sequentially.

2.2 Generating neighbours

We define two local search criteria for finding neighbours: changing vehicle and changing task order.

- Changing the vehicle. We randomly pick up a vehicle with tasks already and remove the first task of it (i.e. remove two actions related to the task). For any other vehicles, we add the task to the front of its action sequence, by inserting two actions related to the task.
- Changing the task order. If a picked vehicle has an action sequence size greater than 2, meaning that it has more than one task, we can change the action order. We can move in two directions of a specific type of action, and this gives us in total 4 combinations.

  (a) Move `Pickup` P forward. When swapping with another `Pickup` action P', we need to make sure that the `Delivery` action of that task D' comes after the current `Pickup` action P.

  (b) Move `Pickup` P backward. When swapping with another `Delivery` action D', we need to make sure that the `Pickup` action of that task P' comes before the current `Pickup` action P.

  (c) Move `Delivery` D forward. Similar to move `Pickup` P backward.

  (d) Move `Delivery` D backward. Similar to move `Pickup` P forward.

  In addition, we need to verify that the load does not exceed the vehicle's capacity.

2.3 Stochastic optimization algorithm

We choose the best neighbour according to a predefined probability, discussed in the section below. We keep track up the global optimal solution searched so far to avoid local minima or fluctuations of stochasticity.

3 Results

3.1 Experiment 1: Model parameters

In our model, the main parameter is the probability $\mathcal{P}$ to choice random neighbors; otherwise, we choose the best neighbor in the process of local choice. This probability balances between exploitation and exploration. When the probability is high, the agent tends to explore different strategies and vice versa.

3.1.1 Setting

We use the default setting, i.e. 30 tasks and 4 agents running the topology of England. Also, in order to search efficiently, we run the training for 50000 iterations, and set the probability $\mathcal{P}$ to choose a random neighbor as $0.0, 0.3, 0.5, 0.7$, and 1.0.

3.1.2 Observations

From the Table 1 and Figure (a), we have the following observations:

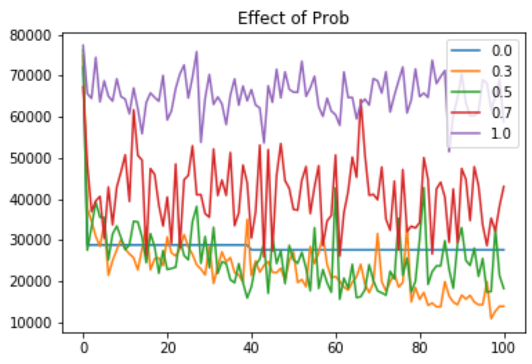
- From Table 1, we find when $\mathcal{P} = 0.3$, the model performs best with the lowest cost and lowest average cost during the training. This perfectly matches with the conclusion in the paper.
- From Table 1, we find that when $\mathcal{P}$ is larger, there is more time for one iteration. The possible explanation is that when the model performs better, it should have fewer neighbors as most of them have been filtered by the constrains.
- From Figure (a), we find $\mathcal{P} = 0.3$ and $\mathcal{P} = 0.5$ perform better than others, and $\mathcal{P} = 0.3$ has a better capacity for convergence.

- $\mathcal{P} = 1.0$ will float at the high cost, while $\mathcal{P} = 0.0$ will be trapped in local minima.

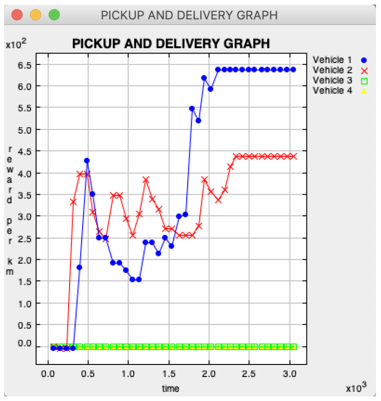
We conclude that during the training process, we find a suitable $\mathcal{P}$ to balance exploitation and exploration: finding the best solution while preventing stuck to local minima.

$\mathcal{P}$	0.0	0.3	0.5	0.7	1.0
Max Cost	72185.5	75017.0	76802.0	69298.5	81447.0
Min Cost	27658.5	10927.0	12757.0	18859.0	42584.0
Average Cost	28158.3	22731.5	25518.1	41197.0	66046.7
Time (ms)	20066	55956	83617	99923	99119

Table 1: Behavior test based on the policy of local choice



(a) Effect of random neighbour choice probability $\mathcal{P}$



(b) 30 tasks on 4 agents, $\mathcal{P} = 0.3$

3.2 Experiment 2: Different configurations

3.2.1 Setting

We run our best model, $\mathcal{P} = 0.3$, on different number of vehicles and tasks for 50000 iterations. We vary the number of vehicles between 3 and 5, and the number of tasks between 20 and 40.

3.2.2 Observations

From Table 2, we have the following observations:

- Task-Time: With the increasing number of tasks, more time will be spent for the planning.
- Vehicle-Cost: For a specific task, the final cost is not closely related to the number of vehicles. We observe that some vehicles do not move, illustrated in Figure (b), which indicates the unfairness among the vehicles.

#Vehicles	3			4			5		
#Tasks	20	30	40	20	30	40	20	30	40
Min Cost	9223.0	16948.5	22616.0	9865.0	15428.0	21653.0	9499.0	17090.0	20321.0
Time (ms)	37664	38267	67695	45481	44282	66181	37714	53690	62305

Table 2: Test for the different configurations