Exercise 4
Implementing a centralized agent

Group 30 : Jinbu Liu, Mengjie Zhao

November 8, 2016

1 Solution Representation

1.1 Variables

We maintain a hashmap to relate vehicles and their tasks. The task set for a specific vehicle is maintained by a linked list. As indicated in the provided documentations, each vehicle can carry several tasks. Hence, each task is represented by a pickup step and a delivery step together. We explore all states under the constraints and find the plan which achieves the lowest cost.

1.2 Constraints

There are three main constraints:

- For a specific task, the delivery step must occur after the pickup step.
- Vehicles can carry several tasks simultaneously. However, the overall weight of carried tasks must be smaller or equal to weight limit of the vehicles.
- All tasks should be pickup and delivered finally.

1.3 Objective function

We try to minimize the overall cost C from all vehicles:

$$C = \sum_{i=1}^{N_T}(dist(t_i, nextTask(t_i)) + length(nextTask(t_i))) \times cost(vehicle(t_i)) + \sum_{k=1}^{N_V}(dist(v_k, nextTask(v_k)) + length(nextTask(v_k))) \times cost(v_k))$$

2 Stochastic optimization

2.1 Initial solution

To generate the initial solution, we assign tasks as equally as possible to every vehicle. First, we compute the remainder of a task id divided by the number of vehicle. If the remainder is 1, the task is assigned to the first vehicle, and if the remainder is 2, the task is assigned to the second vehicle, and so on. Assigning a task to a vehicle means that this vehicle will pickup the task and then deliver it instantly. Using above method, we get the initial solution.

2.2 Generating neighbors

There are two methods to generate neighbor solutions. The first way is to change the delivery vehicle of a task. First, we randomly pick a vehicle and choose a task which is assigned to this vehicle. Second, we remove this task from the task list of the vehicle and assign it to another vehicle. Because delivery of each task contains a pickup step and a delivery step, we insert these two steps into the task list of another vehicle at all possible positions which leads to many neighbor solutions. Finally, we check whether the vehicle is overweight after assigning (a) new task to it and remove all overweight solutions.

The second method is to change the order of steps of a vehicle. First, we randomly pick a vehicle and choose a task, which means we get the pickup and delivery steps of this task. Second, we get other tasks assigned to the same vehicle one by one, and when we get one, we exchange the order of these two tasks to get a neighbor solution, that means we exchange their pickup steps and delivery steps respectively. Using this method, we can obtain many neighbor solutions but we only save not over-weighted solutions.

2.3 Stochastic optimization algorithm

In our algorithm, we have two probabilistic thresholds, namely P_{low} and P_{high}. After generating the initial solution, we generate a random number between 0 and 1. If this number is smaller than P_{low}, we generate the neighbor solutions of the current solution, and use the solution with the minimal cost among neighbors to substitute the current solution. If the random number is higher than P_{low} and smaller than P_{high}, we do nothing. Otherwise, we randomly choose a random neighbor solution to substitute the current one. Another important mechanism in our algorithm is that because the algorithm easily falls into local minimum, we force the algorithm to choose a neighbor solution to substitute the one if the algorithm has stayed in the local minimum for several search iterations. With the search procedure goes on, we always save optimal solution, that is if the cost of current solution is smaller that the optimal solution, we update the optimal solution. At the end of the algorithm, the optimal solution is returned.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting and Observations

In this experiment, we use the England topology and have 4 vehicles and 30 tasks. We change the low threshold from 0.1 to 0.5 and the high threshold from 0.5 to 1.

According to following Table 1, when we keep the high threshold unchanged and only change the low threshold, better solution returns with the low threshold increasing. When we keep the low threshold unchanged and only change the high threshold, better solution returns with the high threshold increasing.

With higher low threshold, the algorithm has higher probability to reach the best neighbor solution, although the algorithm is more likely to fall into the local minimum, we have introduced our technique in section 2.3 which is used to avoid the local minimum. So we are more likely to obtain a better solution at the end. With higher high threshold, the algorithm has lower probability to reach a random neighbor solution. Although this random exploration can be used to avoid falling in to local minimums, it also can interrupt the search process to the optimal solution. So the higher high threshold, the lower probability of the happening of this interruption, which may lead to a better solution.

Table 1: Overall costs among different parameters

P_{low} \ P_{high}	0.5	0.7	0.9	1
0.1	37063	31714	21145	16867
0.3	23921	20022	16752	14020
0.5	21006	18557	15823	13814

3.2 Experiment 2: Different configurations

In this section we carried out many experiments with various parameter settings over our constructed model.

3.2.1 Settings and Observations

In this experiment, the number of tasks varies from 20 to 40 while the number of vehicles takes 1, 2, 4 and 6 respectively. The resting configurations remain the same to that in previous sections. In addition, the two probabilistic thresholds are $P_{low} = 0.5, P_{high} = 1$. Following table displays our experimental results. The cost for provided default configuration ($numTask = 30, numVehicle = 4$) is 14567 and time consumption is 12.65 seconds.

Table 2: Overall costs among different configurations

numVechicles \ numTasks	20	30	40
1	12184	19846	26708
2	10199	15426	19663
4	9344	14567	20275
6	9344	14279	17864

The experimental results meet our expectations. For fixed number of vehicles, larger number of tasks results in higher cost. Since the overall traveled distance for vehicles is longer. For fixed number of tasks, higher number of vehicles results in lower overall cost. This result can be analyzed as following: for a specific vehicle, if there is no other vehicle, the only option it can do to minimize cost is to make order changes among its own task list. However, when there exits other vehicles, this vehicle can choose to change vehicle of tasks. As a result, more states are available, hence it is possible to reach a lower cost.

The optimal plan does not guarantee fairness. In our model, all tasks are finished by only one or two vehicles in most cases, meaning that it is unfair. Following table shows time consumption of our model (in seconds):

Table 3: Time consumption among different configurations

numVechicles \ numTasks	20	30	40
1	3.94	13.27	35.10
2	4.10	13.64	34.62
4	4.29	12.62	30.23
6	3.73	12.73	24.53

It can be observed that the complexity increases significantly with the increase number of tasks. However, the number of vehicles does not introduces significant differences.