Exercise 4
Implementing a centralized agent

Group №22: Dubuis Samuel, Facklam Olivér

November 3, 2020

1 Solution Representation

1.1 Variables

Our constraint problem variables are almost identical to the ones presented in the handout. These variables are represented by a class `Variables` which contains maps of

- `actions` : A map from vehicle to their ordered list of actions. The actions are either "pickup" or "deliver".
- `vehicles` : A map from the actions to their attributed vehicle.
- `timing` : Used to replace the search in the list of actions, it is a direct access to pickup time and delivery time of a task.

1.2 Constraints

These are the constraints that need to be verified on the variables for the plan to be correct:

- Tasks are picked up and delivered exactly once. This constraint is ensured by two checks:
  1. Each task is picked up and delivered at least once: this is done in `checkDelivery()`.
  2. Each task is handled at most once. The functions `checkVehicles` and `checkTiming` verify that each action taken by the vehicles has a correspond entry in the `vehicles` and `timing` maps. Since these maps can only contain a single entry per action, each action is carried out at most once.
- Tasks must be delivered after being picked up: this is ensured by `checkOrder`.
- Pickup and delivery must be done on the same vehicle: checked by `checkVehicle`.
- Vehicle capacity cannot be exceeded at any time: guaranteed by `checkPossibleWeight`.
- Time starts at 1 and increments by 1 at every action: verified by `checkTiming`.

1.3 Objective function

The goal of the company is to maximize its total profit over its whole plan. We immediately see that for a fixed task set, the sum of rewards is constant. Thus maximizing the profit is equivalent to minimizing the plan's cost. This is the same reasoning as in the paper in the handout.

In our algorithm, we are minimizing the objective function corresponding to the sum of costs of each vehicle's plan.

2 Stochastic optimization

2.1 Initial solution

The initial solution is generated by assigning the tasks uniformly at random among the vehicles with sufficient capacity to carry them. The pickup action and the delivery action for these tasks are then appended to the chosen vehicle's plan.

2.2 Generating neighbours

Neighbors are generated using two different operations. In both cases, we copy the variables and only make changes to said copy, either by modifying the order of the actions in the plan of a vehicle, or by changing the vehicle that is assigned to a task.

Swapping two actions in the plan of a vehicle: We choose uniformly at random a vehicle that already has actions in its plan. We then choose two random actions from this vehicle and check if the constraints of weight and order still hold if we swap them. If not, we retry the process.

Changing the vehicle to which a task is assigned: We choose uniformly at random a task already assigned to a vehicle, then choose uniformly at random a different vehicle with sufficient capacity. We remove the action from the first vehicle and add it to the second one. Inserting the new actions in the second vehicle's plan can either be done at the end, or at a random position respecting the capacity constraints.

2.3 Stochastic optimization algorithm

Our stochastic optimization algorithm roughly follows the outline of the stochastic local search presented in the handout paper, with a few particularities.

The search is conducted by doing a parametrizable number of random restarts from a fresh, initial solution. The available planning time is split evenly between each of these iterations.

Each "restart" begins by generating an initial solution as described above. Then during its allotted time, it will generate new neighbors from the current solution, select the best neighbor, and update the current solution to said neighbor with a certain probability.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

The goal of this first experiment is to observe the impact of the major model parameters on the optimization performance. We consider the default topology and configuration, with 30 tasks across England (random seed 12345). We use 4 vehicles and a reasonable 60s plan timeout.

The main parameters we are analyzing here are: `RESTART_NUMBER` (number of random restarts during optimization) and `CHANGE_PROBABILITY` (the probability of moving to the best neighbor at each iteration).

3.1.2 Observations

Table 1 shows the resulting cost of the plan, depending on the two model parameters we are analyzing. We can immediately note that the values are all roughly in the same range, between 11000 and 15500.

For the probability of change, there doesn't seem to be a clear tendency. Still, we can conclude that extreme numbers seem less effective, and anything between 0.5 and 0.8 performs relatively well.

A high restart number (50) leaves little time per restart, so each individual iteration is generally less good. However the large number of trials with high randomization still allows this technique to have similar, or even better results overall, compared to a lower restart number.

		RESTART_NUMBER		
		1	5	50
CHANGE_PROBABILITY	0.1	12971	12095	13701
	0.3	15132	13562	12548
	0.5	12186	12932	11332
	0.7	13015	13146	11332
	0.8	11160	12186	12743
	0.9	13145	12743	12624
	1	13375	12320	12639

Table 1: Total cost of plan, as a function of restart count and probability of change

3.2 Experiment 2: Different configurations

3.2.1 Setting

In this experiment we run different configurations of the environment and interpret the results. First we try to increase the number of tasks with respect to the basic setup. In the next sub-experiment, the number of vehicles is increased compared to the basic setup. These vehicles are separated from others by at least one town. The final experiment is with default number of vehicles and tasks, but we reduce the time allowed for planning.

3.2.2 Observations

Number of tasks	30	60	90
Final cost	11918.0	21699.0	31586.0
Vehicle 1	4	38	44
Vehicle 2	30	20	28
Vehicle 3	20	44	54
Vehicle 4	6	18	54

Table 2: Number of actions per vehicle and final cost for a different number of task

Number of vehicle	4	5	6
Final cost	11918.0	12977.5	12805.0
Vehicle 1	4	0	4
Vehicle 2	30	4	4
Vehicle 3	20	20	20
Vehicle 4	6	6	6
Vehicle 5	-	30	26
Vehicle 6	-	-	0

Table 3: Cost and timing for different number of vehicle

Time allowed to plan in ms	300000	200000	100000
Time taken to generate plan in ms	272691	181775	90894
Final cost	11918.0	11721.0	13374.5

Table 4: Time taken to generate plans in millisecond depending on time allowed

As we can see in table 2, augmenting the number of task (doubling it and tripling it) nearly doubles and triples its cost. It's worth noting however that there always seems to be a vehicle doing less actions.

In table 3 we see that adding vehicle 5 to Liverpool completely shuts down another vehicle, vehicle 1 from Newcastle. Adding a last vehicle 6 in Sheffield seems completely useless. We can interpret by saying that having more vehicles in the experiment might not always be useful.

Finally, in the last table 4, we see that giving less time to the algorithm to plan its course of action doesn't influence the final cost too much.