Excercise 4
Implementing a centralized agent

Group №24: Quentin Juppet, Andrea Piccione

November 5, 2019

1 Solution Representation

1.1 Variables

We use 3 types of variables:

- *nextState* - an array of $N_S + N_V$ variables. The array contains one variable for every existing state (existing task associated with an action (pickup or delivery)) and one variable for every existing vehicle. One variable from the *nextState* can take as a value another state or the value $NULL$ with the following semantics:
  - if $nextState(s_i) = s_j$ it means that some vehicle will operation $action(s_j)$ on $task(s_j)$ immediately after operating $action(s_i)$ on $task(s_i)$;
  - if $nextState(v_i) = s_j$ it means that the vehicle first operate $action(s_j)$ on $task(s_j)$;
  - if $nextState(s_i) = NULL$ it means that the vehicle that manage state s_i does not have any other next states;
  - if $nextState(v_i) = NULL$ it means that the vehicle v_i does not have any next states;
- *time* - an array of N_S variables. The array contains one variable for every state. We therefore have:
  - if $nextState(v_i) = s_j$ the state s_j is the first to be handle: $time(s_j) = 1$
  - if $nextState(s_i) = s_j$ the state s_i is handled immediately after state s_j by some vehicle, and therefore, $time(s_j) = time(s_i) + 1$;
- *vehicle* - a redundant array of N_S variables, one variable for each state. One variable from the array can take as a value the code of the vehicle that delivers the corresponding state such that:
  - if $nextState(v_i) = s_j$ the state s_j is the first to be handled by the vehicle v_i and therefore $vehicle(s_j) = v_i$;
  - if $nextState(s_i) = s_j$ the state s_j is handled immediately after s_i by some vehicle and therefore $vehicle(s_j) = vehicle(s_i)$

1.2 Constraints

1. $nextState(s) \neq s$: the state handled after some state s cannot be the same state;
2. $nextState(v_i) = s_j \implies time(s_j) = 1$: already explained;
3. $nextState(s_i) = s_j \implies time(s_j) = time(s_i) + 1$: already explained;

4. $nextState(v_i) = s_j \implies vehicle(s_j) = v_i$: already explained;
5. $nextState(s_i) = s_j \implies vehicle(s_j) = vehicle(s_i)$: already explained;
6. all task must be delivered and picked up: the set of values of the variables in the *nextState* array must be equal to the set of state S plus N_V times the value $NULL$;
7. $task(s_i) = task(s_j) \wedge action(s_i) = pickup \wedge action(s_j) = deliver \implies time(s_i) > time(s_j)$: the pickup state of some task must happen before the delivery state of the same task;
8. $task(s_i) = task(s_j) \implies vehicle(s_i) = vehicle(s_j)$: the two states of some task are handled by the same vehicle;
9. the capacity of a vehicle cannot be exceeded at any time;

1.3 Objective function

Our objective function aims to minimize the cost of travelling to deliver all the tasks. Therefore, for each vehicle, the cost is defined as the distance travelled by the vehicle multiplied by the cost per kilometer.

2 Stochastic optimization

2.1 Initial solution

We generate the initial solution by assigning all the possible states to the biggest vehicle (the one with the largest capacity). In order to generate all the possible states, we go through all the tasks and compute the two possible states for that task (pickup and delivery) and assign them to the biggest vehicle.

2.2 Generating neighbours

We use two local operators in our current solution to find the neighbours:

- *Changing vehicle*: take the two states of a task from one vehicle and put them at some defined position in the state list of another vehicle.
- *Changing task order*: change the order of two state in the state list of a vehicle.

In each iteration, we choose one vehicle at random and perform local operation on this vehicle to compute the neighbour solutions. For the *Changing vehicle* operator we also select a random task and try all possible positions in the other vehicle state list.

2.3 Stochastic optimization algorithm

The optimization algorithm used is mostly similar to the one described in the reference paper. The only difference is that, instead of tasks, we work with states because we need to allow vehicles to carry more than one task: a vehicle has a set of possible next states and we apply the operations *Changing task order* and *Changing vehicle* to each vehicle and its next states. *Changing vehicle* is a bit different since we try every possible position in the other vehicle and pick a random state (instead of the first one).

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

We analyze a different set of values of both the maximum number of iterations to apply to the algorithm and the probability to return the best solution according to the objective function instead of the current solution. For this experiment, we used the default values and the England topology with 30 number of tasks and 4 vehicles.

3.1.2 Observations

As we can see from Table 1, the time taken to compute the best solution is higher with more iterations. On the other hand, by looking at the optimal cost, having more iterations usually increase the quality of the solution, even though sometimes results are worse with 10000 than with 100000 iterations. Regarding the probability to return the best solution, high values increase the chance to get to a local minima and not find the optimal solution. Indeed, medium-range values are usually better and give a better result.

Probability	1000 iter.		10000 iter.		100000 iter.	
	plan time	best cost	plan time	best cost	plan time	best cost
0.2	890	21579	7750	14091	78081	16604
0.4	975	24683	7224	16845	72821	14905
0.6	940	20484	7261	13781	69969	14091
0.8	869	22570	8098	15346	80777	14627

Table 1: Plan computation time and optimal costs for different values of probabilities and iterations.

3.2 Experiment 2: Different configurations

3.2.1 Setting

We analyze different configurations by varying the number of tasks and vehicles in the environment. We left all the other parameters with the values described in the previous experiment plus maximum iterations=10000 and probability=0.5.

Num. of vehicles	10 tasks		30 tasks		50 tasks	
	plan time	best cost	plan time	best cost	plan time	best cost
1	517	9549	4735	15786	15304	26417
5	1054	10145	9515	16114	25354	26475
10	1781	9196	15528	15911	38692	25816

Table 2: Plan computation time and optimal costs for different number of agents and tasks.

3.2.2 Observations

The results on Table 2 clearly show that the algorithm finds an optimal solutions regardless of the number of vehicles. This happens because most of the times only one vehicle is used to carry all the tasks: this makes sense because one vehicle can potentially deliver and cover more cities when travelling around the topology. Moreover, our initial solution has an impact since we initially assign all the tasks (states) to the biggest vehicle. Obviously, the time taken to compute the optimal plan increases when there are either more tasks or more vehicles because more iterations are required to find the optimal solution.