Excercise 4
Implementing a centralized agent

Group №4: Bruno Wicht, Adrian Valente

November 7, 2017

1 Solution Representation

1.1 Variables

Let us first define a notion that we use in our solution: we call a TaskAction an action that is either a pickup or a delivery. A TaskAction is thus defined by its corresponding task, and a boolean specifying if it is a pickup or not. Obviously, if there are N_T tasks there are $2N_T$ TaskActions.
We have used the following variables:

- An array *nextAction* of $N_V + 2N_T$ entries (one for each vehicle, and one for each TaskAction) mapping from a TaskAction to the next or to $NULL$, and from a vehicle to its first TaskAction.
- An array *time* of $2N_T$ entries mapping from each TaskAction to the position in which it is treated by its vehicle.
- An array *vehicle* of $2N_T$, mapping from each TaskAction to the vehicle that treats it.
- Two arrays *deliver* and *pickup* of N_T entries each mapping from a task to the TaskActions corresponding to its pickup and delivery.

1.2 Constraints

The constraints are the same as those specified in the slides, to which we add the following constraint (expressing the fact that a pickup should happen before a delivery, and that both actions must be done by the same vehicle):

$$\begin{aligned}&(a_j = deliver(t_i)) \wedge (a_k = pickup(t_i)) \\ &\Rightarrow (time(a_j) > time(a_k)) \wedge (vehicle(a_j) = vehicle(a_k))\end{aligned} \tag{1}$$

1.3 Objective function

The function that we optimise is the total cost of a solution, that is the sum of the distances that each vehicle should travel during its plan:

$$C = \sum_{i=1}^{2*N_T} (dist(a_i, nextAction(a_i)) \cdot cost(vehicle(a_i)) + \sum_{i=1}^{N_V} (dist(initialCity(v_i), nextAction(v_i)) \cdot cost(vehicle(a_i))$$

2 Stochastic optimization

2.1 Initial solution

We have first tried to do as in the slides and give all tasks to the first vehicle for the inital solution. However, we found that it was not a good starting point, since it didn't lead to a very good solution. Instead, we start by a random solution: we randomly assign each task to a vehicle, and each vehicle carries the tasks that have been assigned to it in order (by delivering a task before picking up another).

2.2 Generating neighbours

In order to generate neighbors of a solution sol, we first randomly select a vehicle v such that $nextAction(v) \neq NULL$, and then apply 2 primitives: $changeVehicle(sol, v)$ and $changeTaskOrder(sol, v)$.
The neighbors returned by $changeVehicle(sol, v)$ are all the solutions where one task carried by v is given to another vehicle (which picks it up and delivers it before carrying out the rest of its plan).
The neighbors returned by $changeTaskOrder(sol, v)$ are all the consistent solutions where two actions in the plan of v are permuted. The primitive tries out all permutations and returns only those who satisfy to the constraint (1).

2.3 Stochastic optimization algorithm

Our stochastic optimization process follows the algorithm specified in the slides, the termination condition being that no improvement is seen among all neighbors generated (no neighbor with a smaller cost than the current cost is found). However, we found that this condition was rather harsh

given that the neighbor generation is stochastic and can result in a low number of neighbors, especially when there are few tasks. As a consequence, we modified the termination condition so that the algorithm stops when no improvement is seen in at most $NMAX$ calls to $generateNeighbors$, $NMAX$ being a fixed parameter.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

We will be using the English topology, with 30 tasks and 4 vehicles (and random seeds 12345 and 54321). The only parameter that we will analyze is $NMAX$, which controls the termination condition of our algorithm.

3.1.2 Observations

Let us be blunt: the idea of calling $generateNeighbors$ several times does not bring much improvement: indeed, with $NMAX = 4$ we observed a total cost of 7234 ± 955 with 35 ± 10 iterations, while with $NMAX = 1$ we observed a total cost of 8359 ± 1349 and 17 ± 6. So there is an improvement if $NMAX$ is increased, but it is a very limited one, given that the computation time is doubled. That makes us think that the problem may not be of finding a minimum, but of not getting stuck in a local minimum, which appears to be the main factor for excessive costs.

3.2 Experiment 2: Different configurations

3.2.1 Setting

We will try our method with the English topology, with 10, 30 and 50 tasks, and 4 and 2 vehicles.

3.2.2 Observations

With 4 vehicles: we observe a typical cost of 3892 ± 568 with 10 tasks (computation time: 36 ± 11 ms), 7703 ± 404 with 30 tasks (311 ± 51 ms) , and 18568 ± 1277 with 50 tasks (538 ± 133 ms). There are huge disparities between the cost of each vehicle, but that is not necessarily a bad thing, since that can also be the case in the optimum. With 2 vehicles, let us simply say that the cost is a little higher.