Excercise 4
Implementing a centralized agent

Group №75: Thomas KIMBLE, Jules AFRESNE

November 5, 2019

1 Solution Representation

1.1 Variables

Our solution representation has the following variables:

- *vehicles*: A list of all of the simulation vehicles.
- *tasks*: A TaskSet of all of the simulation tasks.
- *worldNbTasks*: An integer equal to the number of tasks in the simulation.
- *worldPlan*: A list of a list to represent each vehicle plan. The sub-list is the pickup or delivery tasks for each vehicle while the main list is of the vehicles themselves. If a pickup task is added, the corresponding delivery task is added $worldNbTasks$ indices after. We get the following for k vehicles and $N_T = worldNbTasks$ tasks:

$$\Big[\ \big[task(1)_{v1}, task(2)_{v1}, ..., task(l)_{v1}, ..., task(1+N_T)_{v1}, task(2+N_T)_{v1}, ..., task(l+N_T)_{v1}\big],$$
$$\big[task(1)_{v2}, task(2)_{v2}, ..., task(m)_{v2}, ..., task(1+N_T)_{v2}, task(2+N_T)_{v2}, ..., task(m+N_T)_{v2}\big],$$
$$... ,$$
$$\big[task(1)_{vk}, task(2)_{vk}, ..., task(n)_{vk}, ..., task(1+N_T)_{vk}, task(2+N_T)_{vk}, ..., task(n+N_T)_{vk}\big]\ \Big]$$

1.2 Constraints

We have the following constraints:

- Two corresponding pickup and delivery tasks must be in the same vehicle.
- A delivery task had to be after it's corresponding pickup task.
- The added weight of pickup tasks before their delivery in a sub-list must be inferior to the vehicles capacity.

1.3 Objective function

First of all we want to minimise the total cost function. For each vehicle we have a distance and a cost per km value. The total cost function is therefore: $distanceCost = \sum_{vehicles} distance \cdot costPerKm$

For our solution also want to minimise the time taken to pickup and deliver all of the tasks. we therefore calculate the time taken for each vehicle to follow its plan and we take the longest time. The time function is therefore: $timeCost = max_{vehicles}(time)$

We therefore combine the two to get the following objective function that we want to minimise. We divide *distanceCost* by the mean of the vehicle speeds (*meanSpeed*) multiplied by the amount of vehicles (N_V) squared to avoid outweighing the *timeCost*. We get the following objective function:

$$Cost = timeCost + distanceCost/(meanSpeed \cdot N_V^2)$$

2 Stochastic optimisation

2.1 Initial solution

We have three possible initial solutions. The first is to assign the tasks randomly to the vehicles while checking constraints to assure a possible solution (case "*randomly*"). The second is to distribute the tasks evenly to the vehicles, here each vehicle will have the same amount of vehicles more or less one (case "*fairlyDistributed*"). The final is to give all of the tasks in to the vehicle with the largest capacity (case "*toTheBiggest*"). We use a switch case to choose, but our program is initialised with the second initial solution (case "*fairlyDistributed*") because it converges faster to an optimal solution.

2.2 Generating neighbours

We generate neighbours two different ways. First of all by moving the first task of a vehicle and its corresponding delivery task to a new vehicle. The second method consists in changing the order of the tasks within a vehicle.

2.3 Stochastic optimisation algorithm

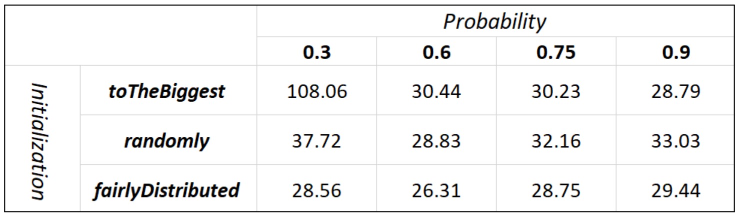
After selecting an initial solution (step 1) we then repeat a new step (step 2) until a termination condition is met. Step 2 has the following sub-steps, where after each one we keep the best solution and use it for the next:
Sub-Step 1: Choose neighbours, This function provides a set of candidate assignment that are close to the current one and could possibly improve it.
Sub-Step 2: Local Choice, where we with probability p we randomly choose one of the best neighbour solutions, and with probability $1 - p$ we return the previous solution. If p is close to 1, the algorithm converges faster but it is easily trapped into a local minimum.
Sub-Step 3: We save this sub solution if it's the best one so far. If there is no improvement of cost after a certain time, we start over with our initial solution and keep the best sub solution by repeating these sub-steps to find a new best sub solution (skip to step 4 without activating termination condition). If there is no improvement after x best sub solutions, we keep the last one and activate the termination condition. We also have a second termination condition if we are approaching a time out.
Sub-Step 4: Repeat until termination condition met. Our last solution is the best one.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

We observe four vehicles in the **England topology** with thirty tasks to deliver. Each vehicle has the same speed and capacity. We will analyse the parameters concerning the probability and the initialisation mode at the same time by taking the cost as a criterion.



		Probability			
		0.3	**0.6**	**0.75**	**0.9**
Initialization	***toTheBiggest***	108.06	30.44	30.23	28.79
	randomly	37.72	28.83	32.16	33.03
	fairlyDistributed	28.56	26.31	28.75	29.44

Figure 1: Cost as a function of the initialisation method and probability.

3.1.2 Observations

The '*fairlyDistributed*' initialisation with a probability of 0.6 is the most conclusive. Nevertheless, we observe that random initialisation is also interesting but that these performances fluctuate due to the random nature of its initial state. The '*toTheBiggest*' initialisation is not very efficient, moreover we noticed that the computation time is very long with this one.

3.2 Experiment 2: Different configurations

3.2.1 Setting

We will use the "*fairlyDistributed*" initialisation in order to have the same initial state for all the experiments. And we set the probability of exploration at 0,6. We will realise the next four experiments in England topology with vehicles having the same speed and capacity:

- 1st: two vehicles and six tasks
- 2nd: two vehicles and forty tasks
- 3rd: four vehicles and six tasks
- 4th: four vehicles and forty tasks

3.2.2 Observations

Here are our results for these experiments:

- 1st: Cost is equal to 27.68 and we also notice that only one vehicle has moved.
- 2nd: Cost is equal to 98.35 and both vehicles has moved but not the same duration because it has less tasks than the other.
- 3rd: Cost is equal to 10.34 and we noticed that one vehicle has not moved. In addition, the tasks were unfairly distributed among the vehicles.
- 4th: Cost is equal to 38.25 and all vehicles have moved.

We notice a factor of about 4 on the cost when we increase from 6 to 40 tasks with the same number of vehicles. This factor is logical because the distance of the vehicles and therefore the travel time has been increased, but it is not proportional to the increase in the number of tasks because the packages have been grouped together in the cities. We can therefore see that vehicles optimise their routes. Moreover through the experiments we noticed that the more the number of tasks and/or vehicles was increased, the longer the calculation time was. This can be explained by the fact that the algorithm is more complex because it has more choices of state. Finally, we noticed that some vehicles do more actions than others, especially in experiment 2. This can be explained by the fact that the tasks are scattered and that the algorithm considers it unnecessary to engage several vehicles when a single vehicle with an optimised route could have an almost equivalent cost. It is necessary to see if we want to privilege the distance cost or to privilege the speed of delivery or both.