Excercise 4
Implementing a centralized agent

Group №45 : Lüscher Jérôme, Delisle Ludovic

November 2, 2020

1 Solution Representation

1.1 Variables

We implemented a class named `NextTasks` that stores the plans we create for each vehicle. The class has three hash-maps as attributes. Initially, the main variable is a hash-map with vehicles as keys and a list of tasks as values. Thus, for a given solution, each vehicle has a sequence of task to accomplish. The problem becomes more complex when we allow the vehicles to carry multiple tasks at the same time. For that, we separate the tasks into two parts: Pickup and Delivery. Both of those two parts extend the class Action that we created. An Action object gives information on the Logist action that it represents and is easier to handle. We end up with a hash-map with vehicles as keys and lists of actions as values. When we find an optimal solution, we convert our action objects into logist actions and we insert move logist actions between actions that aren't taking place in the same city. The third attribute of the `NextTasks` class is a map that links the tasks with their corresponding actions.

1.2 Constraints

While searching for an optimal solution, a lot of potential solutions must be discarded because they do not generate a possible plan. Those are the solutions where the capacity of an agent is exceeded at one time during the simulation and the solutions where a Delivery action of a given task comes after the Pickup action of the same task. The constraints are set up to filter out those solutions.

1.3 Objective function

What our algorithm tries to optimize is the total cost for accomplishing all the tasks. It can be calculated by summing all the displacements of the vehicles and multiplying it by their respective cost per kilometer.

2 Stochastic optimization

2.1 Initial solution

The initial solution is generated by assigning the tasks randomly to each vehicle, without repetition and then to shuffle their order once they have been assigned. Thus, each vehicle has a random set of tasks in a list with a random order, the tasks are uniformly distributed amongst the vehicles. Then we separate the tasks in two parts: Pickup and Delivery. We end up with a hash-map with vehicles as keys and lists of actions as values where Pickup actions are directly followed by their corresponding Delivery actions.

2.2 Generating neighbours

We defined that the solutions neighboring a given solutions were the solutions that could be reached by applying only one transformation on the given solution. A transformation can either be transferring a task from an vehicle to another or swapping the order in the list of two actions in the action list of a given vehicle. This is for neighbors that are one transformation away but one can also be interested by neighbors that are n transformations away. For that, we generate the neighbors of the neighbors until we are n step away from the initial solution.

2.3 Stochastic optimization algorithm

From the initial solution, we generate all the neighbors that are one transformation away(step size 1) and keep the solution with the lowest cost. Then, from that solution, we do the same. We keep doing it until there is no neighbor with a lower cost. It means that our algorithm is stuck in a local minimum. To get out of that minimum, we search a better solution among the neighbours that are 2 transformations away(step size 2). If a better solution is found, we continue the search with size 1 steps until it is stuck in a local minimum again. If no better solution is found, we increase the step size by one and search again. We decided on a threshold beyond which the step size won't increase. When the threshold is reached and no better solution is found, we save that best solution and generate a new random initial solution which maybe will bring the search to a better solution than the previous initial solution. When using the random restart, we have to set a time limit for the algorithm to search.

3 Results

3.1 Experiment 1: Model parameters

When using the default number of tasks of 30, the maximal step size that we are able to use is 2. Beyond that threshold, the number of generated neighbors is simply too big and the algorithm cannot finish the search. When we reduce the number of tasks, we can increase the maximal step size to 3. Thus, the value of the threshold highly depends on the performance of the computer and the settings of the experiment, but the highest possible threshold is optimal. Moreover, as we assign the tasks randomly among the vehicles when generating a fresh new solution, thus there isn't any needed parameter for that part, except the seed. Finally, when using the random restart, we set a time limit of 60 seconds.

3.1.1 Setting

In order to observe the effect of the step size up to a larger threshold, we reduced the number of tasks from 30 to 20. Regarding the other settings, we kept the default ones, that is the England topology, a task weight of 3, a number of 4 vehicles with a capacity of 30.

3.1.2 Observations

We observe on figure 1 that increasing the maximal step size has a increases the efficiency of the vehicles considerably. The total cost of the plan is greatly reduced which means that the total distance that the vehicles travel to deliver all the tasks is reduced proportionally. We also observe that the efficiency when the random restart is enabled is overall better than when it isn't enabled and that the gap in total cost between the two lines gets smaller as the step size gets bigger. These results are expected because the algorithm without random restart is the same as the algorithm with the random restart except that it stops after being stuck in a local minimum while the other one continues and tries its luck from a new fresh starting solution. Thus it cannot give a better solution. However, we can see that for a step size 3, the final cost is the same for the two versions. This is caused by the fact that generating so many neighbors is very time consuming, thus the algorithm with random restarts doesn't have time to do many restarts before its time is up, thus it's chances of finding a better solutions are decreased. Also, generating neighbors using multiple transformations may also enable the solution to evade a local minimum, instead of getting stuck.

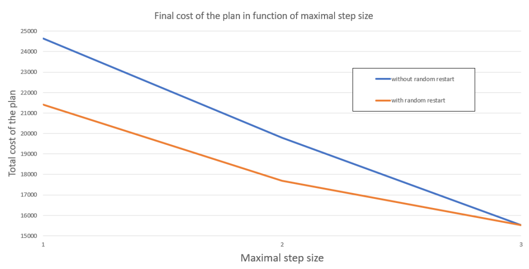

Figure 1: Total cost for delivering all tasks ($n\_task = 20$) for different step sizes, comparing the algorithm with and without random restarts.

3.2 Experiment 2: Different configurations

3.2.1 Setting

In the second experiment, we further reduced the number of tasks to 6, changed the topology to Switzerland and added another vehicle. this is the same settings as for the last exercise where we compared 5 deliberative agents and concluded that they weren't organized. This will show us whether or not they have improved since the last project.

3.2.2 Observations

Compared to last time, we first observe that not only there is no vehicle that has a negative reward/km but also that the overall reward/km is much higher. The most efficient vehicle has a score of ∼1100 reward/km with the centralized algorithm, compared to ∼147 reward/km with the heuristic Astar algorithm. Indeed the vehicles stop directly after delivering their last task, even if there are still available tasks and the vehicles aren't competing for the tasks, thus they do not do any useless moves. Furthermore, we also see that if the efficiency is better when giving a vehicle no task and giving another vehicle multiple tasks, the vehicle with no task will stay idle while the other vehicle does all the work. Finally, due to the low number of tasks, the complexity of the algorithm was reduced, so we were able to increase the maximal step size to 4. However, his increase was not as big as we expected, because increasing the number of vehicles also increases the complexity of the algorithm.

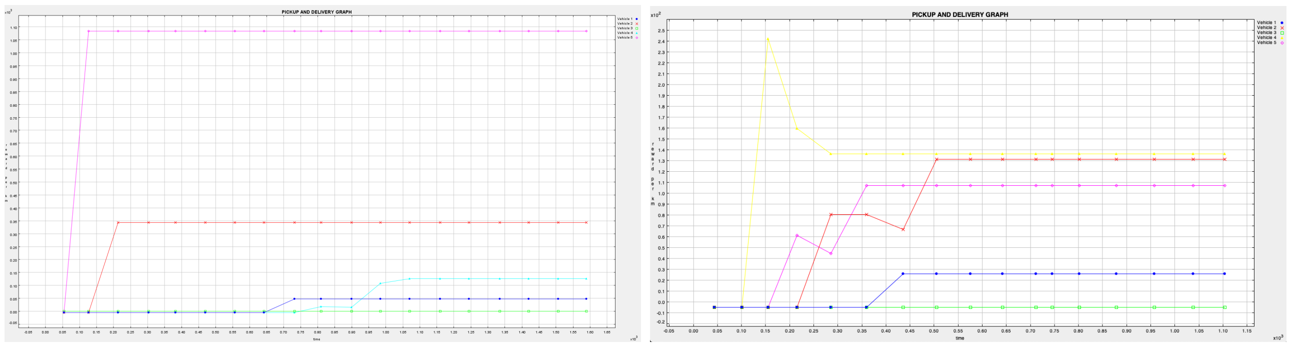

Figure 2: Profit per km plotted against epochs for 5 agents. Left: centralized algorithm described above with step size 4. Right: deliberative agents using heuristic Astar algorithm.