Excercise 4
Implementing a centralized agent

Group №34: Carl Hatoum, Raphaël Linsen

November 3, 2020

1 Solution Representation

1.1 Variables

We defined a **MyAction** class, that represents a task and the nature of the associated action : either pick-up or delivery. A solution consist of each vehicle and its sequence of **MyAction** to be performed.

1.2 Constraints

Two functions are used verify the constraints:

- **checkCapacity**: In our implementation, a vehicle can carry multiple tasks at a time. This function verifies that the total weight of simultaneously carried tasks do not exceed the vehicle's capacity.
- **checkOrder**: For each task, we check that the "pickup" action precedes the "delivery" action.

The other constraints of the CSP are always satisfied since every new neighbour created satisfies them implicitly.

1.3 Objective function

The cost function is the sum of, for each vehicle, the total distance traveled by the vehicle multiplied by its cost per Km. Since every vehicle is at a different position, we expect the algorithm to minimize this cost function by assigning each tasks to the vehicle with the optimal position.

2 Stochastic optimization

2.1 Initial solution

The initial solutions consists of splitting the tasks evenly among all the vehicles. We then assigning once at a time the "pickup" then the "delivery" action for each task assigned. If a vehicle has a lower capacity than than the task's weight, the task is given to the largest vehicle instead.

2.2 Generating neighbours

A vehicle with tasks is chosen randomly, then the neighbours are generate in two steps:

- First, we generate the neighbours where a task of the vehicle is given to another vehicle, for each task. The pickup action and the delivery action are both added to the new vehicle, at every possible positions possible (one neighbour for each).

- Secondly, we generate the neighbours where an action in the vehicle has been swapped with another in the same vehicle, for every possible combination.

Every time, before adding the neighbour, we check whether it verifies the capacity and order constraints.

The point of checking many different combination every step is to allow the algorithm to escape more easily from local minima, albeit at the cost of time per step. For example, considering the move between vehicles, if the action was always added as the first action for the vehicle, it would often be reverted back in the next steps since it increases the cost greatly. Putting the action at the best time increases the likelihood of improving the plan long term.

2.3 Stochastic optimization algorithm

At every step, the neighbours of the current solution are generated and one of them is chosen according to the parameter p:

- With a probability p, a random neighbour is chosen among the ones with the lowest cost. ($\rightarrow$ exploitation)
- With a probability $1 - p$, a random neighbour is chosen. ($\rightarrow$ exploration)

The solution with the lowest cost is stored at all times, and at every step it is updated if a better one is found. The algorithm is set to run for a fixed time duration, after which it converts the best solution the a list of plans usable by Logist.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

The model only has a single parameter, p. Its effect has been quantified on figure 1.

3.1.2 Observations

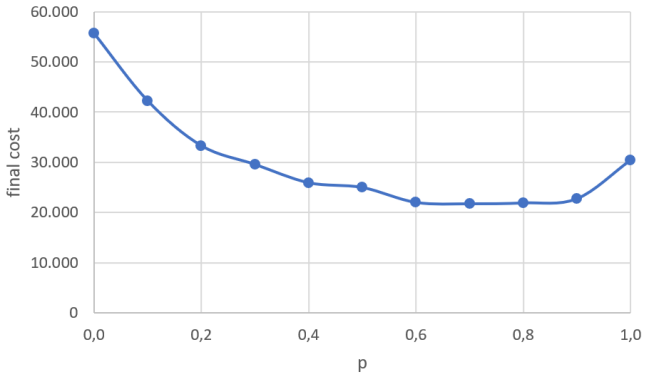
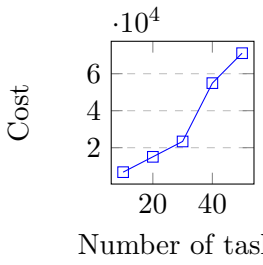
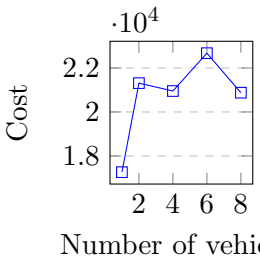


Figure 1: Final cost as a function of p, averaged over 5 different runs (4 agents, 30 tasks, 5 seconds plan)

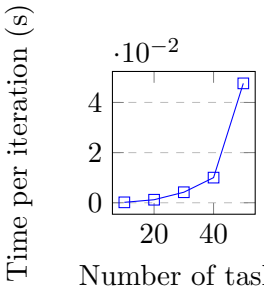
The chosen value was $p = 0.7$. For lower values, the cost decreases thanks to the faster convergence since the best neighbour is taken more often. Beyond 0.9, the algorithm tends to get stuck in local minima, and thus stops improving the solution after a certain point. Indeed, in order to escape a local minimum, the algorithm may require a number of random steps, which can only be realistically achieved if p is not too close to 1.



(a) Cost over number of tasks



(b) Cost over number of vehicles



(c) Complexity over number of tasks

Figure 2

3.2 Experiment 2: Different configurations

3.2.1 Setting

We computed the cost of the optimal solution and the time per iteration performed within the time limit, with 3 vehicles and varying the number of tasks. We computed the cost of the optimal solution, with 30 tasks and varying the number of vehicles. For the two plots, all the remaining settings of the simulation are the default ones.

3.2.2 Observations

One can notice on figure 2a that the more tasks we have, the more the total optimal cost increase. Also, the time per iterations performed in the algorithm increase (fig 2c). Indeed, for the stochastic local search, we generate the neighbors by trying all the possible combinations as previously described. Thus, the problem grows in complexity as the number of tasks increase. The complexity seems to be exponential.

However, the cost do not significantly change with the number of vehicle. Indeed, in the optimal found solution, we notice that the tasks are unevenly distributed across the vehicles : in our example, with 4 vehicles and 30 tasks, vehicle 0 has one task, vehicle 3 the remaining 29, and the others carry 0 tasks.

This makes sense since time is not taken into account, and once a vehicle go through the majority of the cities, it does not cost much to add new actions since they are likely to be on the way of the current path. This is why assigning all the tasks to one vehicle is often a very good strategy.

When tweaking with more parameters, it was observed that the tasks tend to all go the vehicle with the highest capacity or with the lowest cost per km, as it should.