Excercise 4
Implementing a centralized agent

Group №11: Maxime Gardoni, Marcel Dubach

December 8, 2020

1 Solution Representation

For the representation of our solution to the problem we use the following set of variables:
$V = \{v_1, \ldots, v_{N_T}\}$ is the set containing all N_T vehicles, and
$S = \{t_{1,pickup}, t_{1,deliver}, \ldots, t_{N_T,pickup}, t_{N_T,deliver}\}$ is the unordered set of actions to pickup or deliver a task t. The contralized solution A will assign to each vehicle v_i an ordered sequence of actions A_{v_i} that it should perform such that the overall cost is minimized. We implemented A as a *HashMap* that returns A_{v_i} for each of the vehicles v_i.

1.1 Variables

In order to formulate our constraint satisfaction problem we will use the following variables:

- A_{v_i} is an ordered set of actions that will be performed by the vehicle v_i
- $time(t_{i,pickup/delivery})$ returns the time (as integer value) at which the action is performed
- $vehicle(t_{i,pickup/delivery})$ returns the vehicle that will perform the action $t_{i,pickup/delivery}$
- $load(v_i, t_{i,pickup/delivery})$ returns the load of the vehicle v_i after performing the respective action
- $capacity(v_i)$ returns the maximum capacity of vehicle v_i

1.2 Constraints

For our implementation we do have the following constraints:

1. All tasks need to be delivered (if this is possible): $A_{tot} = A_{v_1} \cup \cdots \cup A_{v_{N_T}}$
2. Each task is only delivered once: $A_{v_i} \cap A_{v_j} = \emptyset, \qquad \forall i, j \in 1, \ldots, N_T$
3. Each task is picked up and delivered by the same vehicle:
   $\text{vehicle}(t_{i,pickup}) = \text{vehicle}(t_{i,delivery}) \qquad \forall i \in 1, \ldots, N_T, \forall i \in 1, \ldots, N_T$
4. Each task is picked up before delivery: $\text{time}(\text{t}_{\text{i,pickup}}) < \text{time}(\text{t}_{\text{i,delivery}})$
5. A vehicle of a company cannot carry more tasks at the same time than its capacity allows:
   $\text{load}(v_i, t_{j,pickup/delivery}) \leq \text{capacity}(v_i), \quad \forall v_i \in V, t_{j,pickup/deliver} \in S$

1.3 Objective function

The objective function that we minimize is given as follows:

$$f(A) = \sum_{i=1}^{N_V} c(v_i) \cdot dist(v_i, A_{v_1}) \tag{1}$$

where $c(v_i)$ is the cost per kilometer of displacement of vehicle v_i and dist(v_i, A_{v_i} is the distance covered by vehicle v_i under the partial plan A_{v_i} carried out by the vehicle v_i (including the displacement from its initial location to the pickup City of the first task).

2 Stochastic optimization

2.1 Initial solution

Since we search for a complete algorithm, it should be able to return a feasible solution to the problem if there exists one. Therefore, we first assign all actions $t_{i,pickup/deliver}$ to the same vehicle v_i, which is chosen to be the one with maximal capacity. The actions are ordered such that for each of the tasks, the delivery is performed directly after the pickup ot the same task. If the problem should be unsolvable, meaning that there is a task that has higher weight than the capacity of v_i, the algorithm will detect this directly at the beginning. In all other cases, the algorithm is able to provide a feasible solution.

2.2 Generating neighbours

In order to generate neighbours to the previous plan, we perform one of the two following changes for each neighbour: (1) *Change vehicle*: we assign the first task of a randomly selected vehicle to one of the other vehicles, (2) *Change task order*: we swap the order of two randomly selected actions in the plan of a randomly selected vehicle. In one iteration of the algorithm we randomly select a vehicle and perform (1) $N_T - 1$ times ans assign the first task to each of the vehicles $v_j \neq v_i$, $\forall j = 1, \ldots, N_T$. (2) is performed the same number of times.

2.3 Stochastic optimization algorithm

Algorithm 1: SLS algorithm for PDP

$A \leftarrow InitialSolution(X, D, C, f)$
while *not converged* **do**
  $A^{old} \leftarrow$A
  $N \leftarrow generateNeighbours(\text{A}^{old}, X, D, C, f)$
  $A \leftarrow localChoice$(N,f)
end
return A

The method *localChoice* update A to a probability of p with the best solution among the set of neighbours or assign A^{old} to A otherwise.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

We launch the optimisation for different hyper parameters p. The experience is done with the topology of the England map with a number of 30 tasks of weight 3 each. The company has 4 vehicles of capacity

30 that have identical cost of transport.

3.1.2 Observations

p	0.2	0.3	0.4	0.5	0.75
cost	19140	18749.0	19056	19121	20160
computation time [s]	89.5	86.2	116.4	134.3	141.9
task distribution	[7,4,15,4]	[3,15,5,7]	[7,5,7,11]	[6,4,13,7]	[11,7,5,7]

Table 1: Results of experience 1. The last column indicates how many tasks have been attributed to the vehicles $\{v_1,v_2,v_3,v_4\}$ respectively,

We can see from 1 that the optimisation finds a solution after a time of about 100s due to our stopping criterion. The task seem to be well distributed among the identical vehicles. Values of p around 0.3 and 0.4 seem to perform well. However the result changes from one run to another and it is judicious to analyse of a high number of runs before conclusion. This is however beyond the scope of this report.

3.2 Experiment 2: Different configurations

3.2.1 Setting

We perform now an experience with the 50 tasks of weight 3, keeping the map of England. The vehicle capacity is set to 20 for each of the vehicles.

3.2.2 Observations

p	0.2	0.3	0.4	0.5	0.75
cost	34021	36970	39049	39201	37820
computation time [s]	221.7	151.9	86.7	190.2	113.35
task distribution	[12,5,27,6]	[0,29,15,6]	[3,29,0,18]	[25,5,13,7]	[5,21,6,18]

Table 2: Results of the second experience.

The cost has now more than doubled compared to the previous experience. This is firstly due to the fact that the vehicles can take less tasks at the same time compared to the first experience, and secondly to the fact that there are noe more tasks. Since the number of tasks has increased, there are more more possible solution combinations, therefore it is likely that the proposed solution is further away from the optimum than in the previous case. Remark that in two of the cases the algorithm proposes a solution such that one of the vehicle does not deliver any task. The cost function seems a bit better for the scenarios where the optimisation run for a longer time. This is due to our stopping criterion: The algorithm stops if there are 10e6 iterations without detection of a better solution than the best currently found.