Excercise 4
Implementing a centralized agent

Group № 23: Alessandro Bianchi, Davide Nanni

November 3, 2020

1 Solution Representation

1.1 Variables

`Solution` ::= ⟨ `List<CentralizedPlan>` ⟩. To describe the solution, it is therefore necessary to define:
`CentralizedPlan` ::= ⟨ `vehicle` , `List<CentralizedAction>` ⟩
`CentralizedAction` ::= ⟨ `ActionType` , `task` ⟩.
`ActionType` can be `Pickup` or `Delivery` . A `CentralizedPlan` defines the actions a single vehicle performs in the plan. A `Solution` is a list of `CentralizedPlan` , one for each vehicle. Representing a plan as a list of `Actions` rather than `Tasks` is necessary, since agents can pickup multiple tasks at the same time and therefore a pickup doesn't have to be immediately followed by a delivery.

1.2 Constraints

A `Solution` is valid if all tasks are delivered and all `CentralizedPlan` s are valid. A `CentralizedPlan` for a vehicle `V` is valid if the following constraints are respected:

1. Each picked up task is eventually delivered
2. No task is delivered unless it was first picked up
3. At each step, the sum of the weights of all picked up but not yet delivered tasks is $\leq$ `V.capacity`

1.3 Objective function

cost(`Solution`) ::= $\sum_i$(cost(`centralizedPlan` $_i$))
cost(`centralizedPlan`) ::= `totalDistance` $\cdot$ `costPerKm`

2 Stochastic optimization

2.1 Initial solution

Two different initialization strategies have been devised:

Random Naive The initial solution is computed by assigning each task to the same random vehicle, pushing the `Pickup` and `Delivery` actions in sequence each time.

Closest vehicle The initial solution is computed by assigning each task to vehicles s.t. distance(`currentCity` , `pickupCity`) is minimized. If the nearest vehicle is too full to pickup the task, the delivery action towards the closest delivery city is added. The task to pickup is subsequently analysed again, as the closest vehicle might have changed.

The trade-offs between these initialization strategies will be further analysed in Experiment 1. It is interesting to notice that `Closest vehicle` strategy returns a much lower cost initial solution than the one yielded by `Random naive` (e.g. avg 41749 vs 68731 for the same setting as in Experiment 1).

2.2 Generating neighbours

In order to generate neighbours, a single vehicle is first randomly selected among those that perform at least one action. Then, two different procedures are used:

1. The first task is popped from the vehicle's `actionList`. Subsequently, `Pickup` and `Delivery` for that task are added to the beginning of the plan of each of the other vehicles. This will generate $N_{vehicles} - 1$ neighbours.
2. For the selected vehicle, each action is moved as far as possible to the left or right of the plan (anticipated or delayed, respectively). Each swap creates a new neighbouring plan. For each action, the process continues until the limit is reached, or the corresponding `Pickup` or `Delivery` is encountered. This method is a way of pruning the space of all permutations of a plan, which would quickly become computationally infeasible in the number of tasks.

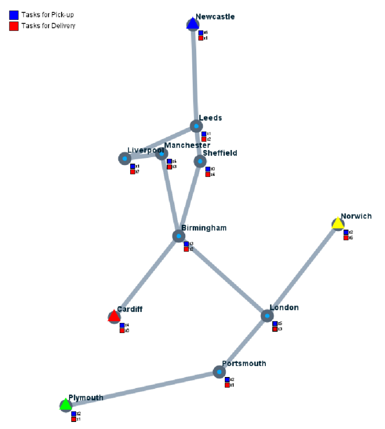
2.3 Stochastic optimization algorithm

At each of the `N_ITER` steps, the stochastic optimization algorithm:

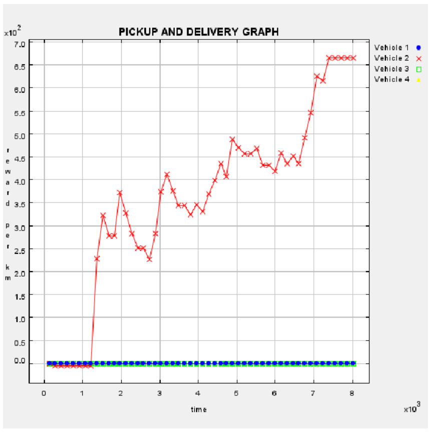
1. with a probability `exploreProb` computes the neighbours of the current solution, and chooses the best of these as the next solution,
2. with a probability 1− `exploreProb` keeps the current solution as the best one for the next iteration.

At the end of every iteration, a `stuck` counter is incremented if no solution that is better than the current optimal one is found. Otherwise, the counter is set to 0. When `stuck` >= `STUCK_LIMIT` the algorithm starts again from the initial solution, as it is likely stuck on a local minimum. The loop continues until the number of iterations is reached or the remaining time for the planning phase is less than the time previously taken by the longest iteration.

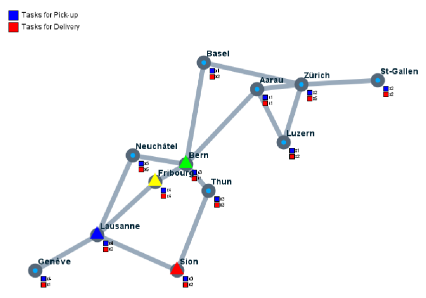
3 Results



(a) GB Topology



(b) Unfair optimality



(c) CH Topology

3.1 Experiment 1: Model parameters

3.1.1 Setting

The experiments were run varying `iterations` number, `stuck-limit` , `explore-prob` and `init-strategy` . All the experiments of this section have been conducted in GB Topology, with 4 agents running at the same time and 30 tasks. The tasks have been seeded in order to make the results obtained among different runs comparable. The following initial parameters have been used: `iterations` = 10000, `stuck-limit` = 1000, `explore-prob` = 1, `init-strategy` = `closest vehicle` . For each experiment, the last parameter set is kept except for the changes identifying the experiment.

3.1.2 Observations

As we can observe from Table 1, the best possible result is obtained by using the Random Naive method with its optimal parameters. However, the Closest vehicle algorithm offers only slightly worse results, but paired with a significantly lower computation time. As a consequence, the most suitable algorithm can be chosen depending on the constraints and priorities of the user.

[*avg of 5*]	Init	20k iter	10k iter, 500 stuck	Expl. P. *0.3*	**Expl. P. *0.5***	**Rand. Naive, 1000 stuck**
Total cost	16473	16018	15882	16537	**15932**	**13205**
Time (ms)	21607	43738	21182	6927	**11046**	**151536**

3.2 Experiment 2: Different configurations

3.2.1 Setting

All the experiments of this section have been conducted in GB Topology with the following parameters: `iterations` = 10000, `stuck-limit` = 500, `explore-prob` = 0.5, `init-strategy` = `closest vehicle` . The number of agent varies from 1 to 4. The number of tasks varies from 10 to 30, with a step of 10. The tasks have been seeded in order to make the results obtained among different runs comparable. The same configurations have been tested on CH Topology. The results appeared to be mostly similar and, as a consequence, it was deemed unnecessary to report them wholly.

3.2.2 Observations

N° agents (30 tasks)	1	2	3	4
Cost	-	14661	14549	15932
Time (ms)	-	62270	21779	11046

N° tasks (4 agents)	10	20	30
Cost	8270	13097	15932
Time (ms)	1199	5041	11046

The results of the experiments can be observed in Table 2 and Table 3. The centralized algorithm is not fair, and does not aim at being so: since only the overall cost (i.e.: the total distance travelled) is minimized, in certain conditions the best plan may involve only one vehicle moving. This is shown in Unfair optimality example. Intuitively, the complexity of the algorithm varies more with the number of tasks than that of vehicles. It is easy to explain the reason for this: while adding one vehicle causes only one more push operation and neighbour state at each step, changing the number of tasks adds significantly more neighbours, as one is generated for each action shifting. A lower number of agents, however, may and will cause a longer list of tasks, as more are given to the same agent: as a consequence, the algorithm may actually take longer with fewer agents. When only one agent is present, it actually becomes unfeasible to compute a centralized plan with 30 or more tasks using the described strategy. It is interesting to notice that in topologies such as GB Topology, the overall cost increases with the number of agents if the `closest vehicle` initialization method is used. This is only due to the topology and the way tasks and vehicles are distributed in the map. A test in the CH Topology, for example, shows a lower cost for a run with 4 agents, compared to one with 3.