# Excercise 4
# Implementing a centralized agent

Group №: 80, Vernet Arthur, Wüst Matthias

November 5, 2019

## 1 Solution Representation

To accommodate the fact that vehicles are able to carry several tasks, it was decided to diverge largely from the original solution proposed in the suggested paper.

### 1.1 Variables

The solution representation consists of a single data structure: a Map mapping each Vehicle to a List of Tasks. The lists are of the following form: $[task_1, task_2, task_1, task_3, ..., task_n]$, which symbolize a sequence of actions, where the first time a Task is listed the Vehicle should pick it up, and the second time the Vehicle should deliver it.

### 1.2 Constraints

Three constraints were established to make sure the data stored in our data structure represents a valid solution.

1. All tasks are assigned, and each task is assigned to a single vehicle.
   Each Task in the domain of Tasks should appear in the List of Tasks of only a single Vehicle.
2. Each task is picked up and delivered only once.
   If the first constraint is respected, it means each Task should appear twice in the List of Tasks of Vehicles. Once as pick-up, and once as delivery.
3. The load of a vehicle never exceeds its capacity.
   Looking at the List of Tasks of each Vehicle, the load is increased every time a Task is added for the first time, and decreased when it is added for the second time. Thus, it is possible to track the total load of the Vehicle.

### 1.3 Objective function

The minimized objective function is trivially the distance covered by each vehicle times the cost per kilometers of the vehicle.

## 2 Stochastic optimization

### 2.1 Initial solution

The initial solution is generated by assigning to each vehicle as many tasks as the vehicle capacity allows it with the same pick-up location that their home city. These tasks are then delivered one-by-one. The remaining tasks are then assigned to a single vehicle, each one being picked up and immediately delivered. By spreading tasks this way, we ensure all vehicles are used, meaning all tasks will be delivered faster. However, since time is not something we're trying to minimize we may end up in sub-optimal minima. Intuitively though, one would agree that it makes sense to take advantage of the fact that some vehicles are already at the location of pick-ups in order to minimize the distance covered by all vehicles.

### 2.2 Generating neighbours

Two functions are used to generate neighbouring solution.

1. *transferingTask*: transfer a Task from one Vehicle to another
   Since a Task appears twice in a List, it is necessary to add it twice to the List of the other Vehicle after having it removed in the first List. The new positions in the List are chosen at random.

2. *changingTaskOrder*: switch the order of two Tasks in the List of Tasks of a Vehicle
   By selecting two Tasks at random in the list, it means either two pick-ups, two deliveries or one pick-up and one delivery may be picked. Their positions are then swapped inside the List.

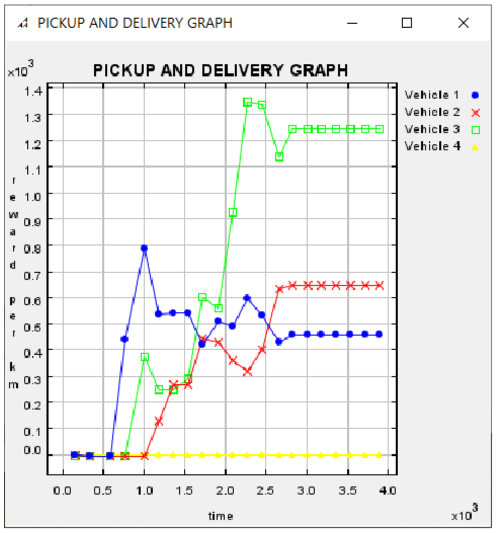
### 2.3 Stochastic optimization algorithm

The algorithm used is a Stochastic Local Search algorithm, which runs for about 5 minutes - the maximum time allowed by the Logist framework. At each iteration, a random vehicle is picked and it generates $N_T \cdot N_V$ (*transferingTask*) + $N_T \cdot (N_T - 1)$ (*changingTaskOrder*) neighbours using the function we defined earlier. Besides the functions used to generate neighbours, it follows the pseudo-code given in the paper as well as the given recommendations. Indeed, even when finding new solutions with a lower cost than the most recently found solution, it will keep $p = 0.4$ of the time the latter in order to avoid getting trapped in local minima.
Additionally, a way of backtracking to older solutions was implemented to try and get out of a local minima. It works as follows: All assignments with a lower cost than the current minimal cost are saved in the list *oldAssignments*. On a current solution and for a number of *maxIterationsOnAssignment* iterations the local search tries to find an assignment with lower cost. If that is not the case, an older solution is taken from the list, stepping back one at a time and the same process happens. When a number of *maxBacktrackIterations* is reached, the time of search is increased, the local search returns back to the currently best assignment, and number of backtrackings is decreased. The idea behind this is to generate multiple solutions which might lead to a good result in the beginning and only in the end to keep searching more extensively at the currently best solutions.

## 3 Results

### 3.1 Experiment 1: Model parameters

The parameters are as described in the section above and the ones changing are *maxIterationsOnAssignment* and *maxBacktrackIterations*. The default values of the topology (England) number of tasks (30) and number of agents (4) were used in this experiment. It is hard to directly compare the effects of changing these values, but nonetheless look below at two runs with different parameters.
Run 1: *maxIterationsOnAssignment* = 20'000, *maxBacktrackIterations* = 50
Run2: *maxIterationsOnAssignment* = 100'000, *maxBacktrackIterations* = 20.



The result of run 1. Run 2 has a very similar outcome

Figure 1: Comparison between different discount factors

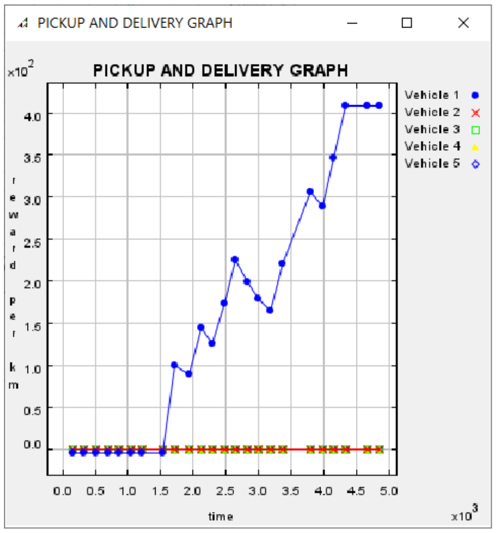
### 3.1.1 Observations

## 3.2 Experiment 2: Different configurations

### 3.2.1 Setting

Topology: England, number tasks: 10, number agents: 5, *maxIterationsOnAssignment* = 20'000, *maxBacktrackIterations* = 50

### 3.2.2 Observations



The result of run 1. Run 2 has a very

Figure 2: Comparison between different discount factors