Excercise 4
Implementing a centralized agent

Group 21 : Martin Vold, Kristoffer Landsnes

November 6, 2018

1 Solution Representation

1.1 Variables

In our solution representation we used three variables.

- The first variable we used is a list of of Plans where Plan number n is this list corresponding to vehicle number n.
- The second variable is a Linked Hash Map with Vehicles as keys and a List of pickup and delivery actions as values, these Lists doesn't contain any move actions. The pickup and delivery actions are encoded as Integers. Every Integer that is smaller than the number of tasks, N_T, and equal or larger than 0 corresponds to the pickup action of the Task with the same ID as the Integer. Integers equal or bigger than N_T and smaller than $2N_T - 1$ is considered as delivery actions for the Task of id equal to the Integer. It is from these Lists of Integers that the Plans for each vehicle is built.
- The third and last variable used in our solution is a Linked Hash Map with Vehicles as keys and a list of Tasks as values. This variable keeps track of which Task a Vehicle is holding for this solution and in the end must deliver.

1.2 Constraints

For a solution to be valid and executable, it must respect the following constraints:

- For all tasks that a vehicle are holding, a deliver action of a Task must never happen before the pickup action for the same Task. If Task t is picked up at time i it must be delivered at time j > i.
- A Task picked up by Vehicle V_i must be delivered by Vehicle V_i.
- Every Vehicle must be able to execute it's sequence of Actions. The sum of weights of Task carried by a Vehicle must never exceed the capacity of the Vehicle.
- Every Task in the topology must be delivered.

1.3 Objective function

The objective function that we want to optimize is the total cost for a Solution as shown in (1).

$$C(S) = \sum_{i=1}^{N_V} (cost(V_i) * length(P_i)) \tag{1}$$

In this equation V_i is Vehicle on index i in the list of Vehicles and P_i is the Plan on index i in the list of Plans. $Cost(v)$ is the Cost Per Km for the given Vehicle and $length(p)$ is the total length of the given Plan. Thus representing the cost for a vehicle to perform the plan.

2 Stochastic optimization

2.1 Initial solution

Because we never know how much time the Logist platform will give us to find the optimal Solution, the initial solution will have to be a valid one. To achieve this we find the Vehicle with the biggest capacity and give it every task. To make sure that this Vehicle never exceeds its capacity it has to deliver one Task after it is picked up and another Task is picked up. Since this vehicle only hold one Task at a time, it respects the constraints. If it doesn't respect the constraint it exists a Task with a weight greater than the highest capacity of our Vehicles and the problem will not be solvable. This initial solution will look like this:

- Sequence of actions for the Vehicle with the largest capacity: $\{0, 0 + N_T, \ldots, N_T - 1, 2N_T - 1\}$
- Sequence of actions for all the other Vehicles: $\{\}$ the empty set

2.2 Generating neighbours

To generate the neighbours for our Solutions we choose one Task t_i at random from a Vehicle v_1. Then we remove t_1 from v_1 and add it to all other Vehicles, v_k. Since we represent the actions for a Task as pickup and deliver, we add a Solution for every possible new combination of the pickup and deliver actions for t_i in the List of these actions for v_k. The deliver task will never be placed before the pickup action.
The other way to generate neighbours is to change the order of the Action plan for a vehicle. A random task t_1 from a vehicle v_1 is chosen. Hereby the Action List of v_1 is iterated to find the index of the Pickup Action and the Deliver Action. All legal combinations is then tried, that is all combinations where the Pickup Action has a lower index than the Deliver Action. All legal possibilities represent a possible solution.
For every neighbor generated, all constraints are valid except constraint number 3 in section 1.2. This constraint is checked when the plans for each Vehicle is generated, see 2.3.

2.3 Stochastic optimization algorithm

The stochastic optimization algorithm is implemented in two steps. Firstly it generates the possible neighbours desired to check for solutions. All neighbours with a valid plan and improved cost are considered a neighbour. If there isn't any possible neighbours the original solution is returned. Afterwards, a LocalChoice function is used to choose a subset of the possible neighbours to search through. With a probability p we return the old solution. With probability $1 - p$ we iterate the neighbour set for the best solution or choose a random solution based on amount of iterations. To avoid choosing random after many iterations, we penalize the random neighbour choice as follows: $1/\sqrt{iterations}$. Thus, when the iterations increase it's less likely to keep choosing random in the neighborhood, but rather explore the best solution. Finally, it's also taken into account that not only solutions with better cost will lead to an optimal solution. We therefore include solutions that are almost as good in the neighbours as well. This is done with a factor of 1.2.

3 Results

3.1 Experiment 1: Model parameters

It was here tested with different probabilities and run-time constraints to assess how it affected the cost of the route. The initial cost was found to be 68731.0.

Probability	**30s**	**45s**	**60s**
p_1=0.2	16305.0	16305.0	16305.0
p_2=0.4	17102.0	14995	13680.0
p_3=0.6	17096.0	17096.0	16470.0

3.1.1 Setting

Tasks: 30. Seed = 12345. Vehicles = 4. Topology: england.xml. The agent centralized-main is used. The parameters changed is the probability, p in LocalChoice.java. The runtime is decided by setting the *timeout-plan* of settings_default.xml.

3.1.2 Observations

We expect the probability to directly effect the solution retrieved by the local search algorithm. Choosing p low will make us look deeper into the solution space, on the contrary a high p value will make us repeatedly be satisfied with the old solution. This is reflected in the table in section 3.1. p_1 can't improve it's solution as much over time as p_2. While p_3 has the same improvement issue since it explores too little.

3.2 Experiment 2: Different configurations

The following experiments have been conducted.
1. 30 tasks for 4 vehicles. Cost = 12530.0
2. 30 tasks for 2 vehicles. Cost = 16860.0
3. 20 tasks for 3 vehicles. Cost = 10249.0
4. 20 tasks for 2 vehicles. Cost = 14383.0
5. 10 tasks for 3 vehicles Cost = 6778.0

3.2.1 Setting

p = 0.4 in LocalChoice.java. Seed = 12345. Topology: england.xml. The agent centralized-main is used. Runtime of 100 s, as described in setting of 3.1.1. If number of vehicles is less than 4, e.g 3; vehicle 1, 2 and 3 has been used in the centralized.xml.

3.2.2 Observations

The most significant observation is that the best solution calculated doesn't require all the vehicles to be in use. Often the best solution only requires 1 or 2 vehicles to act. Since we don't have any direct time requirements in regard of delivering tasks, this doesn't break with any requirements. Even though the time to perform the plan might be slower. An important point is that the COP is minimizing equation (1). When trying to "force" more vehicles into action we changed a requirement when choosing neighbours. By always choosing the vehicle with most tasks assigned, and sending its tasks to other vehicles. This did not improve the cost of the solution, and hence was removed.
As expected increasing the amount of vehicles decreased the total cost. This gives a larger solutions space to explore and hence more possible solutions to the COP.