Excercise 4
Implementing a centralized agent

Group №41: Mohammadreza Ebrahimi, Salar Rahimi

November 3, 2020

1 Solution Representation

1.1 Variables

Vehicles is one of the important variables of our problem. Moreover, To adapt the given algorithm for carrying multiple task at the same time, we arrange the tasks as pickup and delivery. This means the *nextTask*, would be combination of both cases. Therefore, one of our variable is *NextTask*, which would provide the next task (delivery or pickup), for a vehicle and a selected task. We also use something similar as ordering of the task delivery or the same time function as the given algorithm. The only difference is this function arrange both delivery and pickup in the same function and let us know what is next sequence of action (delivery or action).

1.2 Constraints

The most intuitive constraint to check is to compare the capacity of the vehicle to the weight of the carrying and picking tasks. The load of the vehicle should never exceed its capacity. Since we change the ordering of the tasks blindly when we generate neighbors, we should check that we are delivering a task that is actually inside our vehicle. Finally, we check whether we have delivered all the given tasks. Checking all of these constraints for each plan ensures the validity of the generated solution.

1.3 Objective function

Coming up with a cost function is quite straightforward. For each vehicle, we scan all the Move actions and add its cost to the current cost of the plan. Then we sum this cost over all the vehicles. By this we penalise the movement of the each vehicle. For measuring cost of the company, we sum the distance of the each action (pickup or delivery) to the next action (pickup or delivery) plus length of the task picked up. Finally we sum this cost over all the vehicles.Finding the best solution means that the cost of the respective plan is minimal (from the list of all the generated neighbors).

2 Stochastic optimization

2.1 Initial solution

For the initial solution, we take a smarter algorithm comparing to that of the given solution. As a reasonable starting point, we assign the tasks to vehicles that are closest to the *PickupCity*. Since the vehicle can carry multiple tasks at the same time, we chose to maximally fill the vehicle by adding all the tasks that fit inside the vehicle. Then, we deliver all these tasks until the vehicle is empty and ready to pickup all the remaining tasks. Indeed we drop the packages as soon as possible by always iterating over the loaded tasks and comparing it to our current city. This initial solution engages all the agents and starts off with a very reasonable plan that potentially reduces the convergence significantly.

2.2 Generating neighbours

To generate neighbors, we randomly select one of the vehicles. One family of neighbors are generated when we reorder all the Pickup and Deliver actions (Move actions are not important) with all the possible permutations (N!). Of course there are lots of plans for the chosen vehicle that are not valid that will be pruned by the constraint check. Another solution family will be generated by passing the first task of the chosen vehicle to all the other vehicles. The vehicles that receive this task will put it in all the possible indices of their action list (from first task to be delivered to the last one) . Since we change the order of the tasks at some point, the first task would be different almost every time. Following this method, we generate all sorts of neighbors and at the same time we don't drastically change our current plan since it might be close to the optimal solution.

2.3 Stochastic optimization algorithm

After initializing our algorithm with method explained. We will create neighbors by both reordering and changing task as discussed in previous section. Then localchoice method is called. This method picks the least costly plan from the generated neighbors and with probability p substitutes with the old plan. In the case, multiple neighbors has the same minimum cost we select one randomly. The same procedure is handled until the Limit for number of iterations are met or timeout occurs, which we would return the TimeOut failsafe solution. The Iteration limit is selected 10000 for 3 vehicles and number of task of 30 and grows with more complexity added to the problem.

3 Results

3.1 Experiment 1: Model parameters

These are just the default parameter values that was given in the beginning. The reason is that it's quite easy to compare it with the initial random agent.

3.1.1 Setting

Topology: england, Number of Tasks: 30 and equal weight of 3 for all the tasks, Number of Vehicles: 4, p=0.4

3.1.2 Observations

Indeed, it's obvious that the achieved plan outperforms the random one by a major margin. We finish delivering all the parcels in 15 seconds with the cost 69130 whereas for the random Agent can finish even in 2 minutes. It's important to mention that one of the vehicles remains at spot since optimality in overall sense imposes that.

3.2 Experiment 2: Different configurations

To determine whether that the algorithm is quite reasonable, we decrease cost/km of two agents. This should cause these two agents to deliver more tasks than others.

3.2.1 Setting

Topology: england, Number of Tasks: 10 and equal weight of 3 for all the tasks, Number of Vehicles: 3, Cost/Km: 1 for one agent and 5 for others, p=0.4.

3.2.2 Observations

As expected, only the agent with low Cost/Km started moving and delivered all the tasks. All the tasks were delivered in 14 seconds with the cost 6132. The time consumed for the generation of the plan was only 18 seconds whereas in the first setup, it took 100 seconds to fully compute the plans. As a very simple calculation, we can say 10 tasks add at least 20 additional actions (in reality is way more because of the moves we have between cities), that only generates 20! permutations instead of 10!. This alone, significantly changes everything.

3.3 Experiment 3: Different configurations

In this experiment, we prove that the fairness indeed always holds. We maintain all the parameters from the last experiment and prove that if we somehow equally distribute all the tasks blindly, we lose a lot in the cost sense. To do so, for generating neighbors, we don't exchange any tasks. As our initial solution enforces, we assign the tasks based on the distance from the pickup city. Then, we only reorder the plan of each vehicle. This way all the vehicles have some tasks.

3.3.1 Setting

Topology: england, Number of Tasks: 10 and equal weight of 3 for all the tasks, Number of Vehicles: 3, Cost/Km: 1 for one agent and 5 for others, p=0.4.

3.3.2 Observations

Here we observed that although all the vehicles were carrying tasks, the final cost increased to 27216 which is more than 4 times of the latest experiment. This experiment further approves the output of the algorithm.

3.4 Experiment 4: Different configurations

In the final experiment, we investigate the role of the parameter p. We set this value to 0.1, 0.4 and 0.9. This change should change the convergence time

3.4.1 Setting

Topology: Switzerland, Number of Tasks: 15 and equal weight of 3 for all the tasks, Number of Vehicles: 2, Cost/Km: 5 for all the agents, p=0.1 and 0.4, and 0.9.

3.4.2 Observations

The effect of parameter p is significant in the convergence of the system. By setting p=0.4, we converge in 50 iterations with cost 49852. For p=0.1 we converge in 80 iterations with no change of cost and finally with $p = 0.9$ we converge in 25 iterations to the same plan. The fact that the cost remained unchanged proves the robustness of our algorithm.