Exercise 4
Implementing a centralized agent

Group №9: Baris Sevilmis, Doga Tekin

October 31, 2020

1 Solution Representation

1.1 Variables

• PDP: This class contains all the required methods of SLS algorithm including all the helper functions.
• TaskAction: This class is used to denote the actions a vehicle will do by keeping a task and whether the vehicle has to pickup or deliver the task.
• Assignment: This class is mainly for the coordinator to distribute tasks among the vehicles by keeping a hashmap of vehicles and their corresponding TaskAction lists.
• Instead of defining variables such as nextTask, vehicle, and time, our variables are just a list of task actions for each vehicle and these lists have all this information. Time is implied by the index of each task action in its list. The vehicle of a task is implied by the list it is located in. Next actions of a task action are implied by the actions following it in its list.

1.2 Constraints

• PDP-checkConstraints(): For a given vehicle and TaskAction list, the vehicle must not be carrying more weight than its capacity at any point in the list. In addition, the pickup action of each task must come before the delivery action of the task in the list. While changing vehicles and changing tasks, this constraint must be checked to avoid any conflicts.
• Additional weight-capacity check: Vehicle capacities and task weights must be checked before assigning new neighbors to avoid conflicts.
• All tasks must be delivered.

1.3 Objective function

The function we want to minimize is the total travel cost of all our vehicles. Mathematically,

$$C = \sum_{i=1}^{N_v} \text{length}(p_i) \times \text{costPerKm}(v_i)$$

where N_v is the number of vehicles, length(p_i) is the length in kilometers of the plan of vehicle v_i. The length of a vehicle's plan is calculated by starting from the vehicle's home city, and successively adding the length of the shortest path to complete the next task action in its plan until all are completed.

2 Stochastic optimization

2.1 Initial solution

We give all the tasks sequentially (pickup T1, deliver T1, pickup T2, deliver T2, ...) to the vehicle that has the highest capacity. This is guaranteed to be a valid initial solution as long as there are no tasks that weigh more than this vehicle's capacity. If there is such a task, the problem is not solvable. We can improve this initial solution by using our neighbor selection to find a better initial ordering of task actions for this vehicle to use.

2.2 Generating neighbours

We use two methods to generate neighbours, changing the vehicle of a task, and changing the order of a task action of a vehicle.
To create a neighbour by changing the vehicle of a task, we remove both the pickup and delivery actions of that task from the vehicle's plan, and we add them to the end of another vehicle that has the capacity to carry the task. We create many neighbours by trying to give each task of a random vehicle to all other vehicles, and each of these count as one neighbour.
To create a neighbour by changing the order of a task action of a vehicle, we try to change the index of that task action to all other possible locations in the agent's plan. We check constraints for the new plan whenever we make a change to make sure the vehicle does not try to deliver a task before it picks it up and to make sure it doesn't try to carry more weight than its capacity. We create many neighbours by trying to change the order of each task action of a random vehicle, and each of these count as one neighbour.

2.3 Stochastic optimization algorithm

Stochastic algorithm is built same way as the provided pseudocode in the homework description. An initial solution is picked by the method provided in Section 2.1. For a specific amount of iterations or until 1 second to timeout, actual local search takes place, in which old assignment is assigned to a new variable and new assignment is chosen (with probability p) among the suitable neighbors. Neighbors are generated as explained in the Section 2.2. Assignment with minimum objective is picked out of all neighbors and new iteration begins. In addition, best assignment seen so far is kept explicitly as well even if the algorithm does not choose it at an iteration.

3 Results

3.1 Experiment 1: Model parameters

Table 1: Model Parameters-Cost Comparison-Experiment 1

Costs	*Iteration: 100*	*Iteration: 1000*	*Iteration: 10000*	*Iteration: 50000*
Prob: 0.1	42358	23362	19779	13169
Prob: 0.4	28773	18241	14908	12955
Prob: 0.8	28773	17710	17710	14070

3.1.1 Setting

We analyze the effect of changing the parameter p and the iteration count of the SLS algorithm. We test it on the England topology, with the default 30 tasks and 4 vehicles. Tasks are generated with seed 12345 and our algorithm's random number generator is seeded with 0.

3.1.2 Observations

First of all, it should be observable that optimal cost decreases with increasing probability until a certain iteration amount. As known, new assignment will be returned with given probability, otherwise old assignment will be chosen. Therefore for small probabilities, generally the old assignment is returned and for this reason, assignments can't be improved quickly. On the other hand for larger probabilities, there is a higher chance for algorithm to improve quickly but it can get stuck at a local minimum. Table 1 demonstrates that for low iteration amounts higher probability returns lower costs as it improves faster. For low amount of iterations, more slow-exploring agents could not have explored enough to find a better minimum. This trend breaks as soon as the iteration amount is sufficient, where cost of probability 0.4 is lower than the cost of probability 0.8 because the fast exploring 0.8 could not leave its local minimum. For 50000 iterations, it could even be observed that the probability 0.1 results in a better cost than probability 0.8, but still the lowest cost is with probability 0.4. Lastly, it is obvious that the cost decreases overall for higher iteration amount.

3.2 Experiment 2: Different configurations

Table 2: Different Configurations-Vehicles & Tasks

Time(ms)	*Tasks: 10*	*Tasks: 30*	*Tasks: 50*
Vehicles: 4	232	13403	86175
Vehicles: 6	275	7168	30077
Vehicles: 8	225	3953	22945

3.2.1 Setting

We actualize all of our tests in the England topology. We set our iteration amount to 1000 for sake of testing, where we tune our vehicle amounts and task amounts.

3.2.2 Observations

One of the first things to notice is the effect of vehicle amount on timing, where the required time reduces with increasing vehicle amount. Main reason is the task ordering within the neighbour selection method. In general case, more vehicles cause tasks to be more distributed among vehicles, which results in shorter task action lists for each vehicle and therefore less new neighbors. Changing tasks between vehicles does not create as much new neigbours as task ordering does due to the task amount being larger and task ordering to have more variability. For the effects of task amounts, it is obvious that more tasks require more time. Additionally, tasks have variability in terms of the task ordering due to their pickup and delivery properties. In other words, although pickup has to be done before delivery for a specific task, tasks are independent from each other, resulting in polynomial increase of task orderings and neighbours where task size increases. This could be thought also as a result of implementation, in which task ordering list has to be traversed with a nested loop and constraints have to be checked, resulting in $\imath(N^4 + \epsilon)$. The number of neighbours found by transferring tasks between vehicles increases linearly with number of vehicles, meaning that its not a similar increase as task order.
In terms of fairness, we observe in most cases that optimal solutions only use one or two vehicles to do most of the work. This is because vehicles have a lot of capacity and a single vehicle can efficiently find routes in the topology to handle many tasks at the same time, so the cost usually does not decrease when we transfer a task from a vehicle to another one (for example that task might already be on the way to another task, so the vehicle does not decrease its cost by giving that task away). We find that more vehicles become active if we decrease their capacities to carry only a single task at a time.