Excercise 4
Implementing a centralized agent

Group №23: Jean-Thomas Furrer, Emily Hentgen

November 5, 2017

1 Solution Representation

1.1 Variables

We use two variables for representing a plan resulting from the Stochastic Local Search (SLS) algorithm:

Map<Vehicle, TaskAction> *vehicleToFirstTaskAction*: maps a vehicle to the first task action it has to perform (necessarily a pick up, may be `null` if the vehicle picks up no task at all)

Map<TaskAction, TaskAction> *taskActionToTaskAction*: maps a task action to the next task action performed by the same vehicle; used jointly with *vehicleToFirstTaskAction* in order to know which vehicle handles which set of tasks

1.2 Constraints

We (explicitly) enforce X constraints:

– The load of each vehicle must not exceed its capacity.

– The delivery of a task cannot happen before the pickup of this same task.

1.3 Objective function

The objective function we optimize is the total travel cost resulting from all pickups and deliveries of all vehicles.

2 Stochastic optimization

2.1 Initial solution

One possible initial solution consists of assigning all tasks to the vehicle with the largest capacity, in any order. This allows to check the problem is actually solvable: if the weight of the heaviest task exceeds the capacity of the largest vehicle, then there is no solution to this pick up and delivery problem.

In *vehicleToFirstTaskAction*, we add an entry mapping the largest vehicle to a any task pickup, and map all the other vehicles to the `null` value. In *taskActionToTaskAction*, we begin by mapping the first task pickup to its corresponding task delivery. We then map this task delivery to any the pickup of any task in the set of remaining tasks, and map the latter task pickup to its corresponding task delivery, and so on. Each time a task is selected, we remove it from the set of remaining tasks: the mapping *taskActionToTaskAction* is complete once the set of remaining tasks is empty.

2.2 Generating neighbours

We use almost the same two transformations as in the Pickup and Delivery Problem where each vehicle is allowed to carry only one task at a time: a neighbour plan can be generated by either swapping the order of two tasks for the same vehicle, or by giving the first task of one vehicle to another.

We firstly generate a potential neighbour plan without enforcing the constraints. It is after it has been generated that we check whether the constraints are respected: if this is the case, we add it to the set of neighbouring plans, otherwise, we discard it.

2.3 Stochastic optimization algorithm

Our stochastic optimization algorithm will stop either because it reaches the timeout limit defined at the beginning of the simulation, or because there was no improvement in the cost of the plan for a certain number of iterations. For selecting the next plan, it uses a local choice function which, depending on a certain probability distribution, returns a neighbouring minimal cost plan, the current plan, or a random plan (which can be a minimal cost plan). This is because systematically returning a minimal cost plan may lead the stochastic local search to get stuck in a local minimum. This also depends on the initial solution defined at the setup.

Algorithm 1 Stochastic Local Search

```
INPUT
    List<Vehicle> vehicles                    ▷ a list of the agent's vehicles
    TaskSet tasks                             ▷ the set of tasks to deliver
OUTPUT
    Plan plan                                 ▷ a (sub)optimal plan

$initialCountdown \leftarrow 10000$
$hasTimedOut \leftarrow False$, $noImprovement \leftarrow False$
$countdown \leftarrow initialCountdown$
$minimumCostAchieved \leftarrow \infty$
$plan \leftarrow selectInitialSolution(tasks)$

while ((not $hasTimedOut$) and (not $noImprovement$)) do
    $previousPlan \leftarrow plan$
    $neighbourPlans \leftarrow chooseNeighbours(previousPlan)$
    $plan \leftarrow localChoice(neighbourPlans)$

    if $duration > timeout$ then
        $timeOut \leftarrow True$
    end if
    if ($plan.cost >= minimumCostAchieved$) then
        $--countdown$
    else
        $countdown \leftarrow initialCountdown$
        $minimumCostAchieved \leftarrow plan.cost$
    end if
    if ($countdown = 0$) then
        $noImprovement \leftarrow True$
    end if
end while
return $plan$
```

3 Results

3.1 Experiment 1: Model parameters

Our SLS algorithm has three parameters: the probability that a minimum cost plan is selected, the probability that a random plan is selected, and the maximum number of iterations without improvement before the search stops.

3.1.1 Setting

We mostly tested the performance of the SLS algorithm for different probability distributions: as long as the number of iterations without improvement is "large enough", its influence on the cost of the plan is going to be limited. We observed that with a value of 10000, the cost of the plan found by the SLS is relatively stable across multiple simulations. For these simulations, we kept the initial seed of 12345.

3.1.2 Observations

number of tasks	20				30			
$p_{minimum\ cost\ plan}$	0.3	0.4	0.5	0.8	0.3	0.4	0.5	0.8
$p_{random\ plan}$	0.6	0.3	0.2	0.1	0.6	0.3	0.2	0.1
$p_{current\ plan}$	0.1	0.3	0.3	0.1	0.1	0.3	0.3	0.1
minimum cost observed	11 321	10 249	11 157	9 943	14 091	15 944	16 282	13 428
execution time (ms)	82 873	142 337	72 218	76 832	303 013	302 217	186 292	179 886
maximum cost observed	15 706	18 839	22 513	15 593	27 194	37 728	19 584	20 392
execution time (ms)	70 200	74 914	71 191	53 445	209 506	249 670	302 029	185 068

Table 1

For all the different steps, we observed that for a larger execution time, the plan cost was not necessarily lower. What is more, a plan having a similar cost to another could at the same time have been found in twice as much time.
For a relatively large probability of picking a minimum cost plan, even though most of the time, the plan found has a rather low cost, some of the other plans turned out to have a quite large cost, which confirms our intuition about the likeliness of getting stuck in a local minimum.
What is interesting as well is that the cost of a plan does not significantly increase from 20 tasks to 30 tasks: the SLS algorithm manages to combine pickups and deliveries so that the cost of a suboptimal plan remains in the same range. The time it takes to find a plan is significantly longer though.
What is more, when the company has to deliver 30 tasks, for most simulations, the search of the plan does not stop because there is no improvement after a given number of iterations, but actually times out (when 303 000 ms is reached).

3.2 Experiment 2: Different configurations

We observed that for a large number of tasks, the SLS tends to assign all tasks or nearly all tasks to a single vehicle (the vehicle that initially gets all the tasks), and one or two to a second vehicle. This may be because there is a much larger number of neighbouring plans where the order of the tasks is changed, that neighbouring plans that give the first task of a vehicle to another one.

3.2.1 Setting

3.2.2 Observations

If we added a time constraint in the simulation - the company's vehicle must deliver the tasks within a given amount of time, and are penalized in they do not - then the corresponding SLS algorithm would most likely tend to assign the tasks to multiple vehicles.