Excercise 4
Implementing a centralized agent

Group №46: Yann Vonlanthen, Loïc Vandenberghe

November 5, 2019

1 Solution Representation

1.1 Variables

- *nextAction* - an array of $2N_T + N_V$ variables. The array contains two variables for every task, and one variable for every vehicle. The semantics are as $nextTask$ described in [1], except that entry t_i is replaced by two entries: $(t_i, pickup)$ and $(t_i, deliver)$.
- *time* - an array of $2N_T$ variables. The array contains one variable for every tuple $(t \in Tasks, a \in \{pickup, delliver\})$. Defined as in [1].
- *vehicle* - an array of $2N_T$ variables, one for every tuple $(t \in Tasks, a \in \{pickup, delliver\})$. Defined as in [1].

1.2 Constraints

Constraints 1 to 5 are the same as in [1], with a tuple (t_i, a) instead of t_i. Constraint 6 verifies that the set of values of the variables in the *nextAction* array must be equal to two times the set of tasks T plus N_V times the value NULL. The following constraints are specific to the PDP where vehicles can carry more than one task:

6. The capacity of a vehicle can never be exceeded: $nextAction((t_i, a)) = (t_j, pickup) \implies load(t_j) + currentLoad(vehicle((t_i, a)), time((t_i, a))) < capacity(vehicle((t_i, a)))$. $currentLoad(v_k, t_k)$ is the sum of all tasks carried by the vehicle v_k at time t_k.
7. $time((t_i, pickup)) < time((t_i, deliver))$
8. $vehicle((t_i, pickup)) = vehicle((t_i, deliver))$

1.3 Objective function

$$C = \sum_{i=1}^{N_T}(dist((t_i, a), nextAction(t_i, a))) * cost(vehicle(t_i) + \sum_{k=1}^{N_V}(dist(v_k, nextAction(v_k))) * cost(v_k)$$

where:

1. $dist((t_i, a), (t_j, a'))$ is the shortest distance between delivery/pickup location of t_i and delivery/pickup location of t_j, according to a and a'.
2. $dist(v_k, (t_i, a))$ is the shortest distance between the home location of vehicle v_k and the pickup location of t_i.

Note that we tested different objective functions such as M and $M + C$, where M is the maximum cost occurred for a single vehicle.

2 Stochastic optimization

2.1 Initial solution

Our initial solution is obtained by assigning every task in sequence to the vehicle that has the highest capacity. We have not observed better results when changing this, since the high initial temperature makes sure that all task get shuffled around at first.

2.2 Generating neighbours

From a given state we select a random vehicles v that contains at least one task and a task t_i where $vehicle(t_i) = v$. Then we generate 2 kind of neighbors:

- We create a neighbor for every vehicle $v' \neq v$ where the neighbor have the following changes $vehicle(t_i) = v'$, $time(t_i, pickup) = 1$ and $time(t_i, deliver) = 2$.
- We compute all possible delivery and pickup time possible for this task on the vehicle v that satisfies the constraints.

2.3 Stochastic optimization algorithm

The optimization algorithm is inspired by [1] and the Simulated Annealing algorithm. A solution is selected at every step and then the minimal cost solution ever found is returned just before timing out. The algorithm needs 4 parameters: $initialTemp$, $finalTemp$, $pChangeVehicle$, where $0 \leq pChangeVehicle \leq 1$, and $returnToMinimaFactor$. We decrease two temperature variables, one linearly ($0 \leq linTemp \leq 1$) and one exponentially ($initialTemp \leq expTemp \leq finalTemp$).

- With a probability $linTemp \times pChangeVehicle$ we do not compute neighbors and return a random neighbor that assigns a task to a different vehicle.
- With a probability $linTemp \times (1 - pChangeVehicle)$ we do not compute neighbors and return a random neighbor that randomly changes a task order.
- With probability $(1 - linTemp)$ we compute the neighbors as described in 2.2 and select the best one.

After selecting neighbor n, having current solution c, if $cost(n) < cost(c)$ we replace c by n. If this is not the case, we replace c by n with probability $e^{-\frac{cost(n)-cost(c)}{expTemp}}$.
The expontential temperature is computed as follows: $expTemp = initialTemp \times (Lambda)^{\frac{currentTime-initialTime}{timeoutTime}}$
Where $LAMBDA = \frac{finalTemp}{initialTemp}$, this is such that in the last iteration $expTemp = finalTemp$.

As a final mechanism we reset the current solution to the best solution found so far, every time $expTemp$ is divided by $returnToMinimaFactor$.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

For this experiment we used the *england.xml* topology with 30 tasks and 4 vehicles. All vehicles have a capacity of 30 and all tasks has a weight of 3, the cost per km is constant for all vehicles. We set the plan timeout to 1 minute. We analyze different values for the three parameters $initialTemp$,$finalTemp$ and $pChangeVehicle$.

3.1.2 Observations

This first table shows that the temperature range and starting point need to be well calibrated in order to have the best exploration vs. exploitation trade-off. This of course also depends on the topology parameters such as costPerKm and rewards per task.

$initialTemp$	$finalTemp$	$pChangeVehicle$	cost
10_000_000	1	0.2	15687
10_000	1000	0.2	12882
100_000	**100**	**0.2**	**11759**

In this second table we show that the ideal value of $pChangeVehicle$ is indeed around 0.2, as never or almost always changing task assignments is less advantageous.

$initialTemp$	$finalTemp$	$pChangeVehicle$	cost
100_000	100	0	17865
100_000	100	0.1	15927
100_000	100	0.3	12071
100_000	100	0.5	12999

3.2 Experiment 2: Different configurations

3.2.1 Setting

For the first table, we try to play with the vehicles capacity and the cost per km of 2 vehicles. We are using topology *switzerland.xml* with 30 tasks where all tasks have a weight 1, the first vehicle has an unlimited capacity (more than the sum of task weights) and a varying cost per km, while the second vehicle has a fix cost per kilometer of 2 units but has different capacities.

3.2.2 Observations

v1 cost per km	v2 capacity	cost	tasks attributed to v1	tasks attributed to v2
5	1	15580	0	30
4	1	15360	27	3
3	1	6420	30	0
3	2	8769	0	30

We observe a drawback of our algorithm, as when a vehicle has a low cost per km, it tends to have all tasks assigned to it, even tough a vehicle with very high capacity would perform much better long-term. So even though increasing the capacity of the second vehicle would decrease the cost of the optimal solution (since it still includes the solution with lower capacity), it would increase the cost of the solution given by the algorithm since it converges faster to a local minima where vehicle 2 has all the tasks.

On the right table, we can see that our algorithm scales depending on the time it has to run. With longer time to execute, the algorithm can get (slightly) better results since it can do more iterations to explores different neighbors and also can do more iterations to converge to a local minima.

n tasks	timeout	cost
100	6s	31410
100	30s	28950
100	300s	28050

The duration of a single step (i.e. it's complexity), is inversely proportional to the square root of the task set, and almost not affected by the number of vehicles.

References

[1] Radu Jurca, Nguyen Quang Huy and Michael Schumacher, *Finding the Optimal Delivery Plan: Model as a Constraint Satisfaction Problem*, Intelligent agents course, EPFL, 2006-2007.