Excercise 4
Implementing a centralized agent

Group №28 : Mateusz Paluchowski, Vincent Smet

November 8, 2017

1 Solution Representation

1.1 Variables

The only change to the variable representation in comparison to the single-task-in-trunk representation is that every task is given an attribute *TimeInTrunk*. This integer represents the amount of pick-up actions that are performed between the pick-up of the package and the delivery (+1).
Note that this implies that the order and thus also the *Time(Task)* refers to the time of pickup of the package and that delivery actions are not callable with the function *Time()*.

For example, we have the following task order:

$$5 - 3 - 6 - 1$$

If the *TimeInTrunk* of package 5 is 2 than package five will be delivered after the pickup of package 6 and before the pickup of package 1. The function *Time()* for package 6 returns 3.

Note that the deliveries in between pickups are not deterministically ordered by the representation of the solution. Instead when calculating the cost of a solution the shortest distance in between pickups is chosen taken in to account the delivery cities that have to be reached.

1.2 Constraints

A single constraint for a viable solution is added, the capacity constraint. At no point in time should the capacity of a vehicle be violated. This constraint is checked by calling the function *CheckIfPossibleSolution()*.

1.3 Objective function

The same objective function was used as in the single-task-in-trunk implementation. However the implementation of the function *distance()* was altered as already mentioned above.
The distance function is called for two tasks and calculates the shortest distance between the pickup cities of these packages, while dropping off all packages to be delivered in between. To get these deliveries a function *getDeliveries()* is called.

2 Stochastic optimization

2.1 Initial solution

In this implementation a initial solution was chosen in which all tasks are distributed evenly across the different vehicles. (All with *TimeInTrunk* initialised to one) Do note that this initial solution is far from optimal and that a more optimal initial solution might deliver better results. For example packages could be initially assigned to the vehicle with its home city closest to the package pickup location.

2.2 Generating neighbours

The procedure of generating neighbours is performed through the same two functions as the single-task-in-trunk implementation, namely *ChangeVehicle()* and *ChangeTaskOrder*.
ChangeVechicle() ejects one random task out of a vehicles taskset and puts it in front of the the taskset of another vehicle (with a *TimeInTrunk* equal to one).
ChangeTaskOrder() changes the order of the pick-up of two packages for a certain vehicle. Afterwards two possible solutions for different values of *TimeInTrunk* are added to neighbours. This change in task order is performed for two randomized index-pairs.

2.3 Stochastic optimization algorithm

The local choice algorithm computes the total cost of every neighbour. With a certain acceptance probability this neighbour is returned, otherwise a random neighbour is returned. The latter option was taken instead of returning the current solution to obtain a more dynamic model that is less prone to fall into local optima.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

Number of vehicles: 4
Number of tasks: 12 (just for the sake of quick calculations)
Change convergence criteria - number of iterations of CSP are in [100, 1000, 10000]
Change probability of acceptance of best neighbor - [5% 95%]

3.1.2 Observations

Iterations & Probability	Vehicles in Move	Total cost of the solution	Execution time of the s
100 & 5%	3	30703.5	0.85
100 & 95%	2	24752.5	0.97
1000 & 5%	2	27757.0	1.18
1000 & 95%	2	23070.0	0.67
10000 & 5%	2	26317.5	1.4
10000 & 95%	2	22595.0	0.95

Table 1: Model Parameters comparison

As one can deduce from the table above, most of the time only two out of 4 available vehicles actually participate in the delivery of the tasks. Moreover, looking at the execution time column, one can also observe that even though both vehicles participated in delivery, one of those vehicles did the majority of the work (the longer the execution took, the more tasks were given to single vehicle). Moreover, number of iterations seems to have rather limited effect on the final cost of the solution, however (unsurprisingly) the probability of acceptance of best neighbor does - we simply pick best neighbors more greedily.

3.2 Experiment 2: Different configurations

3.2.1 Setting

Number of Vehicles: 2 - 4 - 6
Number of iterations: 10000
Number of Tasks: [4, 12, 20]
Probability of acceptance: 85%

3.2.2 Observations

Vehicles & Tasks	Vehicles in Move	Computation time [ms]	Total cost / No. Tasks
2 & 4	1	480	1434.25
2 & 12	2	2817	1698.04
2 & 20	2	37883	1658.4
4 & 4	2	523	1294.25
4 & 12	2	2219	1754.95
4 & 20	-	-	-
6 & 4	2	570	1348.25
6 & 12	3	3659	1564.04
6 & 20	-	-	-

Table 2: Configurations comparison

As one would expect increasing vehicle and tasks space increases complexity and affects computational time with cases where solution is not found due to timeout. Here yet again only few vehicles are favored and given some tasks to distribute. Last but not least, unsurprisingly the total cost of solution divided by number of tasks ratio is increasing with amount of tasks simply due to more amount of moves that needs to be performed in order to deliver all tasks.