Exercise 4
Implementing a centralized agent

Group №70: Jelena Banjac, Stratos Triantafyllou

November 5, 2019

1 Solution Representation

- $V = \{v_1, v_2, ..., v_{N_V}\}$ be the set of vehicles owned by the company, and available for delivering the tasks; N_V is the total number of vehicles in this set;
- $T = \{t_{1,pickup}, t_{1,deliver}, t_{2,pickup}, t_{2,deliver}, ..., t_{N_T,pickup}, t_{N_T,deliver}\}$ be the set of pickup/delivery (PD) tasks, where each task is paired with the corresponding type of the action (pickup or delivery); $2N_T$ is the total number of tasks in this set.

1.1 Variables

We use the following variables to define a constraint satisfaction problem (CSP) that finds the optimal plan for the company, the meaning of variables is similar to the ones from the instructions paper:

- *nextTask* - an array of $2N_T + N_V$ variables. The array contains two variables for every existing task (i.e. for task t there is t_{pickup}, $t_{deliver}$), and one variable for every existing vehicle:

$$\begin{aligned} nextTask = [&nextTask(t_{1,pickup}), ..., nextTask(t_{N_T,pickup}),\\ &nextTask(t_{1,deliver}), ..., nextTask(t_{N_T,deliver}),\\ &nextTask(v_1), nextTask(v_2), ..., nextTask(v_{N_V})]; \end{aligned} \tag{1}$$

  One variable from the *nextTask* array can take as a value another PD task, or the value NULL:

$$nextTask(x) \epsilon \{t_{1,pickup}, t_{1,deliver}, ..., t_{N_T,deliver}, NULL\}; \tag{2}$$

- *time* - an array of $2N_T$ variables. The array contains two variables for every existing task (i.e. for task t there is t_{pickup}, $t_{deliver}$).

$$\begin{aligned} time = [&time(t_{1,pickup}), ..., time(t_{N_T,pickup}),\\ &time(t_{1,deliver}), ..., time(t_{N_T,deliver})]; \end{aligned} \tag{3}$$

  One variable from the *time* array can take an integer value specifying the actions sequence number of the task in the plan of a certain vehicle:

$$time(x) \epsilon \{1, 2, ..., 2N_T\}; \tag{4}$$

- *vehicle* - an array of $2N_T$ variables. The array contains two variables for every existing task (i.e. for task t there is t_{pickup}, $t_{deliver}$).

$$\begin{aligned} vehicle = [&vehicle(t_{1,pickup}), ..., vehicle(t_{N_T,pickup}),\\ &vehicle(t_{1,deliver}), ..., vehicle(t_{N_T,deliver})]; \end{aligned} \tag{5}$$

One variable from the *vehicle* array can take as a value the code of the vehicle that picksup/delivers the corresponding task:

$$vehicle(x) \epsilon \{v_1, v_2, ..., v_{N_V}\}; \tag{6}$$

1.2 Constraints

1. $nextTask(t_{pickup}) \neq t_{pickup}$; $nextTask(t_{deliver}) \neq t_{deliver}$
2. $nextTask(v_k) \neq t_{j,deliver}$ the first task of any vehicle cannot have action deliver;
3. $nextTask(v_k) = t_{j,pickup} \implies time(t_{j,pickup}) = 1$
4. $time(t_{pickup}) < time(t_{deliver})$
5. $nextTask(t_{i,pickup}) = t_{i,deliver} \implies time(t_{i,deliver}) = time(t_{i,pickup}) + 1$;
   $nextTask(t_{i,type}) = t_{j,pickup} \implies time(t_{j,pickup}) = time(t_{i,type}) + 1$, $type \epsilon \{pickup, deliver\}$;
   $nextTask(t_{i,type}) = t_{j,deliver} \implies time(t_{j,deliver}) = time(t_{i,type}) + 1$;
6. $nextTask(v_k) = t_{j,pickup} \implies vehicle(t_{j,pickup} = v_k)$;
7. $nextTask(t_{i,pickup}) = t_{i,deliver} \implies vehicle(t_{i,deliver}) = vehicle(t_{i,pickup})$;
   $nextTask(t_{i,type}) = t_{j,pickup} \implies vehicle(t_{j,pickup}) = vehicle(t_{i,pickup})$;
   $nextTask(t_{i,type}) = t_{j,deliver} \implies vehicle(t_{j,deliver}) = vehicle(t_{i,pickup})$;
8. all tasks must be delivered: the set of values of the variables in the $nextTask$ array must be equal to the set of PD tasks T plus N_V times the value NULL;
9. the capacity of a vehicle cannot be exceeded: if $load(t_{i,pickup}) > currentAvailableSpace(v_k) \implies vehicle(t_{i,pickup}) \neq v_k$.

1.3 Objective function

The goal of the project was to minimise a total cost for transporting all parcels. The following function is used to calculate the total cost:

$$C = \sum_{i=1}^{N_v} totalDistance(v_i) \cdot costPerKm(v_i) \tag{7}$$

2 Stochastic optimization

2.1 Initial solution

For a simple initial solution, we give all the tasks to the biggest vehicle. If there exist some task that do not fit the vehicle, then the problem is unsolvable. That means: $vehicle(t) = v_k$, where t is any task in the system, and v_k is the biggest vehicle. $time(t_{1,pickup}) = 1, time(t_{1,deliver}) = 2, ..., time(t_{N_T,pickup}) = 2N_T - 1, time(t_{N_T,deliver}) = 2N_T$ so the tasks are immediately delivered after the pickup. $nextTask(v_k) = t_{1,pickup}$, $nextTask(t_{pickup}) = t_{deliver}$, $nextTask(t_{N_T,deliver}) = NULL$ when the last task is delivered, the vehicle finished the transport.

2.2 Generating neighbours

We use two following operators for finding the neighbors of the current solution: (1) *Changing vehicle*: take the first task from one vehicle and give it to another, we iterate through all available vehicles and randomly choose the one that will receive new task; (2) *Changing task order*: change the order of two tasks in the task list of a vehicle.

2.3 Stochastic optimization algorithm

In our CSP encoding we have: $X = nextTask, time, vehicle$ is a set of variables ($n = 6N_T + N_V$); D is set of domains specified above, C is set of constraints specified above, and f is the objective function specified above. A sketch of SLS algorithm that was followed during the implementation is given bellow:

Algorithm 1 SLS algorithm for COP

```
1: procedure SLS(X, D, C, f)
2:     A ← SelectInitialSolution(X, D, C, f)
3:     repeat
4:         A^old ← A
5:         N ← ChooseNeighbors(A^old, X, D, C, f)
6:         A ← LocalChoice(N, f)
7:     until termination condition met
8:     return A
9: end procedure
```

3 Results

3.1 Experiment 1: Model parameters

Our model accepts two parameters. First, we have the number of iterations that define the termination condition (e.g. how many times the algorithm will be executed). We also have the probability p which we have introduced in the context of diversification: at each iteration, we select the optimal neighbor solution with probability p, otherwise we opt for a random neighbor regardless of how much it improves the original solution. This, along with the rollback strategy, helps with avoiding falling into local optima.

For experiments with both parameters, we maintain the initial configuration: Topology: England; Number of tasks: 30, Number of vehicles: 4

Maximum number of iterations We ran the model varying the number of iterations from 100 up to 100 000. As expected, the more the iterations, the better the quality of the solution. For 100 iterations, we get a cost of 25 141, and this goes as down as 15 056 for 100 000 iterations. This can be explained by the fact that the model has more opportunities to explore the solution space and come up with a better solution.

Probability of choosing the best neighbor as local choice We varied the diversification probability p between 0 (the model only selects a random neighbor at each iteration) and 1 (the model only opts for the best one). For 500 iterations of the algorithm, we got our best solution by balancing p between 0.3 and 0.5. However, as we increase the number of iterations to 10 000, the model has additional time to explore different candidates, and the diversification scheme looks to be more beneficial with p close to 0.

3.2 Experiment 2: Different configurations

Size of the task set A greater number of tasks results in a more complex execution plan and a higher solution cost, as expected.

Number of vehicles Our model finds an optimal solution of 15 056 that employs two vehicles to carry out the tasks. As such, the number of vehicles practically does not impact the performance of the model.