# Excercise 4
# Implementing a centralized agent

Group №: 272257, 272709

November 3, 2020

## 1 Solution Representation

We formalize the pickup and delivery problem as a *constraint optimization problem*. The *COP* is formally a tuple $< X, D, C, f >$ where $X$ is a set of variables describing a plan, $D$ the domain of the variables $x_i \in X$, $C$ a set of constraints that a plan needs to fulfill and $f$ a cost function that we are trying to minimize. A plan $P$ is given by an assignment of the variables in $X$ (where the constraints in $C$ are satisfied). We call $N_v$ the number of vehicles and $N_t$ the number of tasks.

### 1.1 Variables

Any plan can be described by a given assignment of the 4 following variables (in out implementation, a plan is represented by the `Solution` class):

1. **nextTask_v** : the $nextTask\_v$ variable is an array of size $N_v$ where each element $nextTask\_v[i]$ represents the first task that the $i^{th}$ vehicle will perform.

   if $nextTask\_v[i] = j$ it means that the $i^{th}$ vehicle will start it's route by delivering the $j^{th}$ task

   if $nextTask\_v[i] = NULL$ it means that the $i^{th}$ vehicle has no task to deliver in the given plan.

2. **nextTask_t** : the $nextTask\_t$ variable is an array of size $2N_t$ where each element $nextTask\_t[i]$ represents the next task that the vehicle that the vehicle will pickup (in this array the pickup and delivery are represented as two distinct array element to allow for multiple tasks to be picked up by the agent)

   if $nextTask\_t[i] = j$ it means that the vehicle that delivered the $i^{th}$ task will deliver the $j^{th}$ task next.

   if $nextTask\_t[i] = NULL$ it means that the vehicle that delivered the $i^{th}$ has no task to deliver next.

3. **time** : the *time* variable is an array of size $2N_t$ where each element $time[i]$ represents the position of the $i^{th}$ task in the plan of the vehicle delivering it in the plan (so if it is the first task delivered by some vehicle we would have $time[i] = 1$, again it is of size $2N_t$ to represent pickup an delivery as distinct actions)

4. **vehicle** : the *vehicle* variable is an array of size $2N_t$ where each elements $vehicle[i]$ describes which vehicle will delivered the $i^{th}$ task

### 1.2 Cost function

Since we are looking for an optimal solution of our Constrained Optimization Problem (in the sense that it minimizes a cost function) we need to define our cost function.

Given :

| | |
|---|---|
| *Distance between two cities $c_1$ and $c_2$:* | $dist(c_1, c_2)$ |
| *Cost per kilometer for a given vehicle:* | $C_{km}(v)$ |

We define the cost function $cost(P)$ as :

$$cost(P) = \overbrace{\sum_{v \in [1 \ldots N_v]} \left[ \mathcal{C}_{km}(v) \cdot \sum_{path} dist(c_{i1}, c_{i2}) \right]}^{\text{where } c_1 \text{ and } c_2 \text{ are two cities on vehicle } v\text{'s path}}$$

### 1.3 Constraints

Not all possible variable assignments correspond to a valid plan, a valid plan is a plan that satisfies all constraints $c \in C$, the constraints are the following:

| | |
|---|---|
| *The next task after a given task t cannot be itself* | $nextTask\_t[t] \neq t$ |
| *The time variable must be coherent* | $nextTask\_v[i] = j \Rightarrow time(j) = 1$ |
| | $nextTask\_t[i] = j \Rightarrow time(j) = time(i) + 1$ |
| *All tasks must be delivered* | $\#$ *NULL values in nextTask_t* |
| | $= \#$ *non-NULL values in nextTask_v* |
| *From the definition of vehicle* | $nextTask\_v(k) = j \Rightarrow vehicle(j) = k$ |
| *From the definitions of nextTask_t and vehicle* | $nextTask\_t[i] = j \Rightarrow vehicle[j] = vehicle[i]$ |
| *From the definitions of nextTask_v and vehicle* | $nextTask\_v[i] = j \Rightarrow vehicle[j] = vehicle[i]$ |
| *A vehicle cannot carry more than it's capacity* | $load(i) > capacity(k) \Rightarrow vehicle(i) \neq k$ |

## 2 Stochastic optimization

Because of the high computational complexity of the problem we search for a solution using a *Stochastic Local Search method (SLS)* that can be described by the following algorithm :

### 2.1 Stochastic optimization algorithm

**Algorithm 1:** Stochastic Local Search for Constrained Optimization Problem

```
S ← GenerateInitialSolution(< X, D, C, f >)
repeat
    A_old ← A
    N ← Succ(A_old, < X, D, C, f >)
    A ← Choice(N, f)
until stable_solution ∨ timeout;
return A
```

This algorithm is implemented in the `plan` function of the `CentralizedAssignement` class.

### 2.2 Initial solution

We initialize our *COP* with a naïve solution to the pickup and delivery problem for multiple agents where we give each agent a random subset of tasks (in such a way that all tasks are handled) in a random order. This initial assignment doesn't account for the possibility of having multiple tasks loaded simultaneously by a single agent, so each pickup is immediately followed by a delivery in the initial plan (which may not be true for all following plans).

### 2.3 Generating neighbors

The set *succ* of neighbors is generated (implementation in the `neighbours` function of class `Solution`) by generating every possible exchange of two tasks in the order of pickup/delivery of tasks and every possible change of assignment of vehicle per task. So each *succ* is a set of size $N_t! + N_v \cdot N_t$.

### 2.4 Choosing neighbors

In order to minimize to cost we choose the neighbor that will replace the current plan in the next iteration according to the following law (let $0 < p < 1$):

**Algorithm 2:** Choice(N,f)

With probability $p$:
**return** *plan* $\in N$ *such that* $f$ *is minimal*
else
**return** *plan* $\in N$ *where plan is randomly selected*

## 3 Results

### 3.1 Experiment 1: Model parameters

#### 3.1.1 Setting

We run a series of simulations for the PDP with a timeout of $30s$ and a varying number of agents (from 1 to 4 agents). We use the seed `12345`. Each agent is given a cost per kilometer of 50.

#### 3.1.2 Observations

We get the following results :

| *1 vehicle* | *2 vehicle* | *3 vehicle* | *4 vehicle* |
|---|---|---|---|
| $cost = 183775$ | $cost = 258665$ | $cost = 259915$ | $cost = 3759$ |

We observe that the total cost does not change linearly with the vehicle number, we believe this is a consequence of the fact that the algorithm performs a local search.

### 3.2 Experiment 2: Different configurations

#### 3.2.1 Per-agent cost and fairness

Running the algorithm with varying per-vehicle cost shows us that for an unbalanced *cost per km* distribution. The system will allocate example, given a configuration of $\mathcal{C}_f(v_1) = 100, \mathcal{C}_f(v_2) = 50, \mathcal{C}_f(v_3) = 30, \mathcal{C}_f(v_4) = 1$ will lead to a completely *unfair* repartition of tasks where the vehicle with cost $\mathcal{C}_f(v_4) = 1$ will carry every single task. It is interesting to note that this is also a consequence of our *cost per km* metric, it would be very different if we optimized over time.