Exercise 4
Implementing a centralized agent

Group №70 : Pierre-Antoine Desplaces, Julien Perrenoud

November 6, 2018

1 Solution Representation

1.1 Variables

In order to simplify the notation later on, we first define a general set of actions $A = P \cup D$ where $P = \{ p_1, ..., p_n \}$ is the set of pick-up actions for each task with id 1 to n, and $D = \{ d_1, ..., d_n \}$ is the set of corresponding delivery actions.

Using this, our set of variables is expressed as $X = X_{\text{time}} \cup X_{\text{vehicle}} \cup X_{\text{prev}} \cup X_{\text{load}}$ where:

- $X_{\text{vehicle}} = \{ \text{vehicle}(a) \mid a \in A \}$ is the vehicle carrying each action a. The domain is the set of vehicle, i.e. $\mathcal{D}(X_{\text{vehicle}}) = V = \{ v_1, ..., v_k \}$.
- $X_{\text{time}} = \{ \text{time}(a) \mid a \in A \}$ is the time of each action a. As there are $2n$ total actions (one pickup and one delivery for each of the n actions), we have $\mathcal{D}(X_{\text{time}}) = \{ 1, ..., 2n \}$
- $X_{\text{prev}} = \{ \text{prev}(a) \mid a \in A \}$ is, for each action a, the action executed right before. If an action is the first one, we will simply use *null*, therefore $\mathcal{D}(X_{\text{prev}}) = A \cup \{ null \}$
- $X_{\text{load}} = \{ \text{load}(a) \mid a \in A \}$ is the vehicle load after performing action a. The domain is $\mathcal{D}(X_{\text{load}}) = \{ 0, ..., \sum_{i=1}^{n} w_i \}$ where $w_i \in W = \{ w_1, ..., w_n \}$ corresponds to the weight of each task.

It is important to note that this is only the mathematical representation of our solution. In code, a lot of the redundancy can be removed and our solution is completely defined via a combination of the X_{vehicle} and X_{prev} variables, represented as a `HashMap<Label, Label> successors`, where `Label` corresponds to the possible values in $A \cup V$.

$$v_1 \rightarrow p_1 \rightarrow p_3 \rightarrow d_3 \rightarrow d_1 \rightarrow done$$
$$v_2 \rightarrow done$$
$$v_3 \rightarrow p_2 \rightarrow d_2 \rightarrow done$$

Figure 1: Lightweight solution representation

```
HashMap<Label, Label> successors {
      v1: p1, p1: p3, p3: d3, d3: d1, d1: null,
      v2: null,
      v3: p2, p2: d2, d2: null
}
```

Figure 2: Equivalent in code

1.2 Constraints

In this problem, are two types of constraints. First, as we "augmented" our minimal representation of the problem (X_{prev}) with additional variables in order to express more complex constraints, we need to ensure that those additional variables are consistent with the representation given by X_{prev}. Then, we can use these additional variables to clearly express the constraints of the system:

1. Consistency of X_{time}, X_{load} and X_{vehicle} with regards to X_{prev}.
   - $\text{prev}(a_i) = a_j \Rightarrow time(a_i) = time(a_j) + 1 \ \ \forall a_i, a_j \in A$
   - $\text{prev}(a) = null \Rightarrow time(a) = 1 \ \ \forall a \in A$
   - $prev(p_i) = null \Rightarrow load(p_i) = w_i \ \ \forall i \in \{ 1, ..., n \}$
   - $prev(p_i) = a \Rightarrow load(p_i) = load(a) + w_i \ \ \forall i \in \{ 1, ..., n \}, a \in A$
   - $prev(d_i) = a \Rightarrow load(d_i) = load(a) - w_i \ \ \forall i \in \{ 1, ..., n \}, a \in A$
   - $prev(a_i) = a_j \Rightarrow vehicle(a_i) = vehicle(a_j) \ \ \forall a_i, a_j \in A$
2. Time must not overlap, except for different vehicles
   - $time(a_i) = time(a_j) \Rightarrow vehicle(a_i) \neq vehicle(a_j) \ \ \forall a_i, a_j \in A, a_i \neq a_j$
3. Delivery must happen after pickup, with same vehicle
   - $time(d_i) \geq time(p_i) \ \ \forall i \in \{ 1, ..., n \}$
   - $vehicle(p_i) = vehicle(d_i) \ \ \forall i \in \{ 1, ..., n \}$
4. Load does not exceed max (c_v is the capacity of a given vehicle $v \in V$)
   - $vehicle(a) = v \Rightarrow load(a) \leq c_v \ \ \forall a \in A, v \in V$

1.3 Objective function

As we required to deliver all tasks, there is no need to include the reward of each individual delivery. Our objective function is therefore only the sum of the fuel cost of each vehicle to perform all the deliveries, and we try to minimize it.

2 Stochastic optimization

2.1 Initial solution

Our initial solution is generated by randomly selecting tasks one at at time and inserting them as first naive delivery of a randomly selected vehicle (with enough capacity to carry the task).

2.2 Generating neighbours

At each iteration, a set of neighbors is generated by randomly selecting a pickup action p_i and moving it somewhere else (including in another vehicle). Each such replacement has up to n corresponding valid assignments - one for each valid position of the corresponding delivery action - which we all generate. We repeat this operation a constant number of times m (with the same action p_i but a different position).

2.3 Stochastic optimization algorithm

After selecting a random initial assignment, our optimization algorithm executes each step by selecting a random set of m neighbors and selecting the lowest cost one. If the neighbor is better than the current assignment, it is selected with probability 1. Otherwise, it is only selected with a fixed probability p.

As soon as the algorithm does not encounter a better local solution for a given number of steps $nIter$, it resets to a new random initial assignment. When the timeout occurs, the algorithm returns the best assignment found overall.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

Here, we run our algorithm with 30 tasks of weight 3 carried by 4 vehicles with capacity 30. Each time, the algorithm is given 30 seconds to find a solution.

3.1.2 Observations

In Figure 3, we see how $p = 0.05$ allows our algorithm to converge towards best solutions while still allowing it to make bad moves sometimes in order to explore potentially better solutions in close neighborhoods (final cost 12804.5). In Figure 4, we see that adding a reset with $nReset = 500$ allows the algorithm to find another better neighborhood faster and thus find a better overall solution (11502.0). Finally, 5 displays how a small neighborhood ($m = 1$) or a value of p too high (not shown here, but the effect is similar) can prevent the algorithm from quickly converging towards good solutions. Finally, it is also important to note that what is considered to be good values of p and m might be different based on the goal. Using $p = 0$ allows us to converge faster towards a good enough approximation, but $p > 0$ might be necessary when hoping to find an optimal solution.

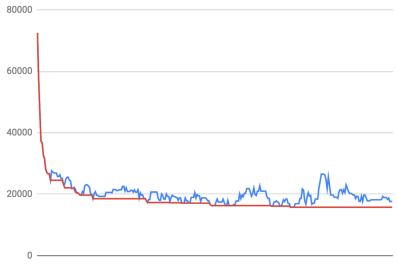

Figure 3: $p = 0.05$, $m = 10$, $nReset = \infty$

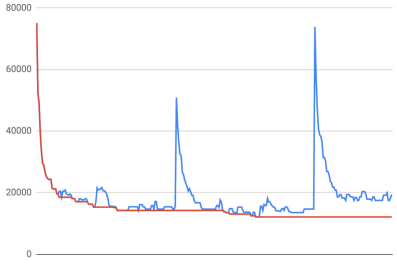

Figure 4: $p = 0.05$, $m = 10$, $nReset = 500$

Figure 5: $p = 0.05$, $m = 1$, $nReset = \infty$

3.2 Experiment 2: Different configurations

3.2.1 Setting

Here, we analyze the distribution of tasks in similar settings as in the first experiment but with different number of tasks and vehicles. The solver ran long enough (300s) to try and find the best possible solution.

3.2.2 Observations

We see in the following table that close-to-optimal solutions often require several idle vehicles. We can explain this by saying that as the only cost considered is the distance travelled, one vehicle could deliver as well as several vehicles by executing their task assignment in sequence (the cost of moving to their home city is minor and can be removed depending on the ordering). More, it can also try and outperform by shuffling their tasks in a smarter way. Here, for large n, we still might not be close to optimal solutions.

vehicles/tasks	30	100	200
4	0/23/7/0	14/16/39/31	66/44/48/42
6	0/21/0/9/0/0	34/17/17/0/32/0	38/35/35/33/36/23
8	0/12/7/0/0/0/11/0	41/0/29/0/0/0/0/30	30/16/39/45/37/33/0/0