

Excercise 4

Implementing a centralized agent

Group №3 : Vincent Petri, Yannick Grimault

November 8, 2016

1 Solution Representation

1.1 Variables

In our implementation, we created a class `CUSTOMACTION` which is a tuple of a task with an action on this task (`pickUp` or not `pickUp`, represented by a boolean). Note that we once again had to implement `HASHCODE` and `EQUALS` to compare 2 similar actions.

This class allowed us to work not on a list of `PickUp`, `Move` and `Deliver` Actions, but on a smaller set of actions where we could access the corresponding task (we can't access the task of a `LOGIST.ACTION`).

In our algorithm, we had to replace the list of plans by a list of list of our custom actions, so it became problematic when we wanted to keep a copy of the currently optimal planification. That's why we had to create the function `COPYPLAN` which makes a deep copy of our planification.

Obviously, we also had to implement a function `LISTTOPLAN` that transforms a list of `CUSTOMACTION` into the corresponding Plan by adding the `Move` actions too.

Note that we also implemented the functions `NEXTACTION` (2 implementations whether we provide a vehicle or an action as an argument), `TIME`, and `VEHICLE`, as described in the assignment description.

1.2 Constraints

Our constraints are the following, and are tested in our function `ISPERMUTABLE`:

- A task has to be picked up before being delivered
- A task can't be picked up if its weight added to the total weight of the tasks carried by the vehicle surpasses the capacity of the said vehicle

1.3 Objective function

The function that we want to optimize is implemented in `TOTALCOST`. It simply computes the total distance travelled by all the vehicles, multiplied by their respective cost. Note that we went for an economic solution, so it might not be the fastest one.

2 Stochastic optimization

2.1 Initial solution

Our initial solution simply consists in the translation of the naive plan in our variable system: the first vehicle picks up and delivers each task one by one.

2.2 Generating neighbours

To generate neighbours, we choose a task at random, then whether we look at its pickup or its delivery. Then we choose at random whether we want to entrust this task to another vehicle or permute it (when possible) with its successor (this verification being done by `ISPERMUTABLE`).

If we chose to entrust the task to another vehicle, we need to move both pickup and delivery actions at the end of the stack of actions of the said vehicle.

2.3 Stochastic optimization algorithm

For our stochastic exploration, we repeat a certain number of times `UPDATE` which “moves” to a random neighbour. We then compute the cost of this neighbour, and if it's less than the cost of the previous optimal solutions, we store a copy of it. At the end of the loop, we simply use the optimal plan stored and compute the corresponding list of plans in terms of `LOGIST.ACTION`.

3 Results

3.1 Experiment 1: Model parameters

3.1.1 Setting

3.1.2 Observations

3.2 Experiment 2: Different configurations

3.2.1 Setting

3.2.2 Observations