Deliberative agents

Jeremy Gotteland & Quentin Praz

October 28, 2015

World representation

State representation

Recall that the agent here refers to a single vehicle. We chose to represent the state of an agent as the following:

- Current city: The city in which the agent is currently located.
- Vehicle: The type of vehicle
- Cost: The cost from initial city to arrive in that state. Computed as the number of kilometers times the cost per kilometers of the vehicle.
- Available tasks: The tasks the agent has not picked up yet
- Carried tasks: The tasks the agent is carrying and has to deliver.
- currentWeight / freeWeight: The sum of the carried tasks.
- List of actions: list of actions necessary to arrive at this state from initial state.

Our *State* object contains a method called 'getNextStates()' that returns the states in which we can arrive from current state. We differentiate three cases:

1. Next step could be to **deliver** any of the task we currently carry. Then our next city would be one the task's delivery city. We would end up with one task less to carry and less weight.

2. Next step could be to **pickup** any of the task we yet have to pickup (available tasks) **given that we have enough free weight to carry that task**. Then our next city would be one of the available task's pickup city. We would end up with one more carried task, one available task less, and more weight.

3. Next step could be to **delivery and pickup** some tasks if the pickup and delivery cities are the same, given that the removal of the carried task gives us enough weight to pickup a new one. This is a mixture of the two previosu scenarios.

We are in a final state when we have no more available tasks to pickup, nor carried tasks to deliver

Heuristic

First, let's define the cost of going from a city to another one as the following (CPK being the Cost per Kilometer of the vehicle):

$$C(c_i, c_j) = D(c_i, c_j) * CPK$$

Our heuristic for A* is the following:
Given a state $\mathbf{S}$ in city $c_{current}$, that can go deliver tasks in a set of city $\mathbf{D}$, and pickup tasks in a set of cities $\mathbf{P}$, their union being the set of next possible cities given state $\mathbf{S}$: $\mathbf{N} = \mathbf{D} \cup \mathbf{P}$ the heuristic H_S is equal to

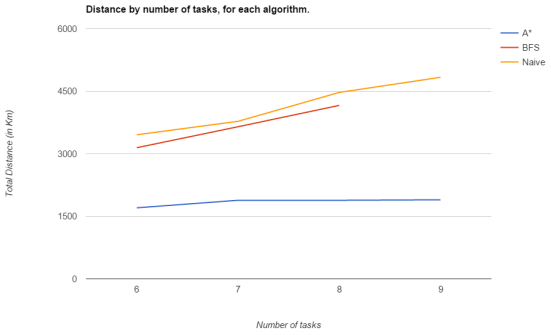
$$H_S = \max(\max(C(c_{current}, c_n)), \max(C(x))), \forall c_n \in \mathbf{N}$$

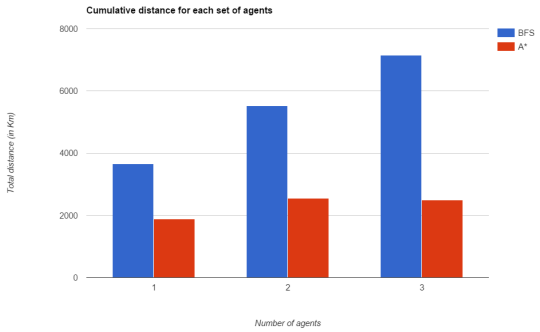
This heuristic makes sense because it does not overestimate the cost: in every case we will have to go through at least one of those paths, so we take the maximum of them all as our heuristic because it will be the minimum distance we will go through.

It is optimal because the choices we make are directed towards taking the shortest paths to the final state where we will have delivered all possible tasks.

Performance Comparisons between BFS and A*

We first ran the simulation with only one agent through both algorithms, as well as the naive algorithm, trying consecutive values for the number of tasks.
We observe that the performance of the BFS algorithm and the naive algorithm are quite close.
Moreover, when we try to compute BFS for more than 8 tasks, we get a Timeout exception from Logist.
A*, as an addition to being quicker, also gives better performance as for the total distance covered.





Then we run the simulation with 1, 2, and 3 agents, for 7 tasks.
As discussed with the TAs, we discovered a bug in the Logist platform, which causes the first action from the agent's plan is **executed before the next agent starts to compute his plan**. Thus, the performance of the following agents is greatly improve as the problem they face is a subset of the initial problem (one less task), and so on for the next agents..

None of those 2 algorithms is suited for cooperation as the agents do not consider each other. However, because of the problem previously stated, the A* algorithm gives very good results as the number of agents increases.

Conclusion

In this exercice, we discovered two main algorithms for implementing deliberative agents: BFS and A*.

We proved by observation that, if the heuristic is carefully chosen, A* outperforms BFS greatly as the number of tasks and agent increases.
This is due to the fact that the heuristic guides the algorithm towards the right direction, whereas BFS's performance is very dependent on the order in which we visit the neighbours of each node.

Using DFS (Depth-first Search) with pruning instead of BFS would have given better performance