

# A reactive agent solution for the Pickup and Delivery problem

Prisca Aeby, Alexis Semple

The task for this lab was to create a deliberative agent for the pickup and delivery problem. A deliberative agent, unlike a reactive one, has complete explicit knowledge of its environment at a given point in time, and can build a plan according to some logical reasoning.

Our task was to define a state representation and implement two algorithms for the computation of a plan for our agent: a BFS and an A\* algorithm for finding a path from the starting state to a finishing state. We took any finishing state to be a state where all the tasks in the world have been delivered.

## 1 State description

We derived the state definition from a description of the state-transitions. Our agent moves from one state to the next by either adding the actions for a pickup or for a delivery to his plan. Hence every state is defined by the remaining tasks in the world and for each one whether its currently being carried by a vehicle or waiting to be picked up.

In other words, our state is a plan which keeps track of where the remaining tasks are in the world.

## 2 Implementation

Our implementation uses java's `PriorityQueue` structure to store the states while computing a plan. The BFS and A\* algorithms interact with the structure in slightly different manners. We used different comparators to allow this.

## BFS

The BFS comparator compares the **depth** of the two plans being compared, i.e. the total number of pickups and deliveries made in a plan. In this way, it will give higher priority to plans who are in lower levels of the state tree, since a new level is reached by making either a pickup or a delivery.

## A\*

The A\* comparator needs to take a heuristic into account in order to determine priority. We defined it as follows:

$$\begin{aligned} h(n) &= \min_i \{dist(p_i, d_i)\}, i \in T, \\ T &= \text{set of remaining tasks,} \\ p_i &= \text{pickup city of a task, } d_i = \text{delivery city of a task,} \\ dist(a, b) &= \text{distance of the shortest path between cities } a \text{ and } b \end{aligned}$$

By defining  $h(n)$  to be the minimum, we ensure that the heuristic is admissible, since it never overestimates the cost to reach the goal. The function then compares two plans by computing for each one the value of  $f(n) = g(n) + h(n)$  where  $g(n)$  is the distance needed to get to the current state (i.e. the `totalDistance()` of the plan)

## 3 Testing

In order to assess the performance of both algorithms we compared them to each other and to the naive-plan algorithm that was already given in the skeleton program. We tested these using different configurations

- CH, 6 tasks, BFS, 1 agent → 66585 states, 2740km for the plan
- CH, 6 tasks, A\*, 1 agent → 3291 states, 1380km for the plan
- CH, 6 tasks, BFS, 2 agents → total distance of 3890km for the plan
- CH, 6 tasks, A\*, 2 agents → total distance of 2120km for the plan
- CH, 6 tasks, BFS, 3 agents → total distance of 4960km for the plan
- CH, 6 tasks, A\*, 3 agents → total distance of 2660km for the plan

- CH, 7 tasks, BFS, 1 agent → doesn't terminate
- CH, 7 tasks, A\*, 1 agent → 235014 states, 1610km for the plan
- CH, 8 tasks, A\*, 1 agent → doesn't terminate
- FR, 6 tasks, BFS, 1 agent → 66585 states, 8709km for the plan
- FR, 6 tasks, A\*, 1 agent → 5208 states, 4341km for the plan

What we can see here first of all is that A\* clearly performs better than BFS, since it finds the optimal solution and does so in less iterations. Note that the naive algorithm gets a total distance of 4420km for a single agent, in Switzerland with 6 tasks, which is considerably worse than BFS. This tendency is consistent when we use two or three agents.

The tests show a rapid state-space explosion when we add tasks to the set. For 7 tasks, BFS can't cope, while A\* has to run 235'014 iterations to find the plan (previously 3291, for 6). For 8 tasks, even A\* can't cope.

The algorithms perform similarly on a different topology. So for France, with 6 tasks, we get a low number of iterations for A\*, and a high one for BFS (identical to that of Switzerland, since the graph will have the same size for the same number of states) and a total distance for A\* which is approximately half that of BFS.

## 4 Conclusion

A deliberative agent is a good solution for a world where there won't be any interference with a computed plan. It's able to compute an optimal solution, using A\* and a simple heuristic. However, for cases where the computation of a plan is costly (i.e. with 7 tasks or more in our implementation), and recomputing the plan could be necessary when executing it (several vehicles picking up and delivering the same task set), this can be very inefficient.