MAS report week 5

Bram Kooiman 11415665
Akash Raj Komarlu Narendra Gupta 11617586
Florian Mohnert 11770929

8 March 2018

1 Activities

This week's goal is to implement a form of a group decision (e.g. whether to add new buses). We make use of the buses' local knowledge on which passengers there is a plan for. A new bus is added if the ratio of people for which there is no plan to the total number of people grows too large. We also updated decision making for routes.

1.1 Adding new buses

On predefined ticks, all buses can assess how many passengers exist in the world and how many of those passengers are accounted for (note that they may still be waiting, but at least a bus is on its way to pick him/her up). If there are too many, the bus votes to add another bus. If the amount of unaccounted-for passengers grows suddenly we add a bigger bus. Later in the day we'll be more hesitant to add new buses. Since the available information and procedure on determining this vote is the same for all buses, the buses will always agree. This removes the need for sending messages. One bus can decide on whether to add a new bus, knowing that its peers always have the same vote. In a way this is a 'perfect representative democracy'; the actions of the leader will always perfectly represent the will of the majority of the population (in this case even the entirety). It is still a group decision because every bus contributes to the information pool.

1.2 Planning a route based on expected utility

We had a few issues with the previous implementation of route planning. If a bus considers a back-and-forth route, say, $2 \rightarrow 3 \rightarrow 2 \rightarrow 3$, it would count the utility to be gained from picking up the people at station 2 and dropping of passengers at station 3 twice, though it can only be performed once. We resolve this by keeping track of the variable would_have; the people that would be on board at some step in the route. The flag "b" signifies that this passenger would

be on board. The flag "d" signifies that this passenger would have already been dropped off.

We test our implementation on a simple test case. The graph is as follows. A csv file with passenger information is included in the hand-in. The bus should decide that for a route of length 2, the maximum utility to be gained is from the route $3 \to 2 \to 4$.

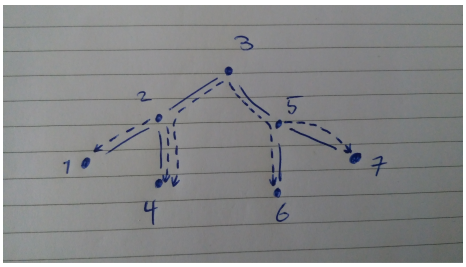

Figure 1: Graph for the test case

Moreover, for a route of length 3, we check if the bus doesn't commit the back-and-forth fallacy for route $2 \to 3 \to 2 \to 3$:

```
(bus 24): "departing for bus stop 2 the bus would have these on board:
            [[0 4 b] [1 6 b]]"
(bus 24): "when we get there, utility would be 1.25 and these would be on board:
            [[0 4 b] [1 6 b] [3 4 b] [2 1 b]]"
(bus 24): "departing for bus stop 3 the bus would have these on board:
            [[0 4 b] [1 6 b] [3 4 b] [2 1 b]]"
(bus 24): "when we get there, utility would be 0 and these would be on board:
            [[0 4 b] [1 6 b] [3 4 b] [2 1 b]]"
(bus 24): "departing for bus stop 2 the bus would have these on board:
            [[0 4 b] [1 6 b] [3 4 b] [2 1 b]]"
(bus 24): "when we get there, utility would be 0 and these would be on board:
            [[0 4 b] [1 6 b] [3 4 b] [2 1 b]]"
```

For another test case, we check if the bus realises that a passenger with the flag "d" cannot be picked up again. In this test case, there is only one passenger wanting to go from 3 to 2.

```
(bus 24): "departing for bus stop 2 the bus would have these on board: [[0 2 b]]"
(bus 24): "when we get there, utility would be 0.08 and these would be on board: [[0 2 d]]"
(bus 24): "departing for bus stop 3 the bus would have these on board: [[0 2 d]]"
(bus 24): "when we get there, utility would be 0 and these would be on board: [[0 2 d]]"
(bus 24): "departing for bus stop 2 the bus would have these on board: [[0 2 d]]"
(bus 24): "when we get there, utility would be 0 and these would be on board: [[0 2 d]]"
```

2 Practical Notes

In order to keep a clear structure of our source code, we determined to add include-files to the top-level file "agents.nls". These are included in the hand-in.

In order to run the test cases, replace the following line in the function "get-daily-ridership-schedule"

```
let file\_name (word "passengers-location\_day" days ".csv")
```

with

```
let file\_name "test\_case1.csv"
```

In order to recreate the output of this report, uncomment the show statements in the functions "actual_utility" and "determine_route". Both functions can be found in the file "determine_route.nls".