Final Multi-Agents System report

Emma Machielse, Federico Tavella, Alessandro Tezza

April 1, 2018

max 10 pages. **readability**; grammar, syntax, scientific style, reference fig and tables. **completeness**; what is done, why and the results, motivation design choice, how does system work

1 Introduction

Public transport is an universal subject of travellers' complaints, targeted at the waiting and travel time. Attempts of transportation companies to reduce costs often have a detrimental effect on travelling time. A network of intelligent autonomous transportation units could optimize for lowering costs and travelling time. To illustrate this, a multi-agents system is created for an intelligent bus transport in Amsterdam. The goal of this system is to transfer passengers as efficient as possible. Several methods stated in game theory [**?**] are implemented to create this system, like communication between buses, coordination and negotiation.

2 Framework

We chose to organise our system using a Belief-Intention architecture, similar to the BDI one [**?**]. In fact, every agent has a set of intentions, i.e., a set of tasks that it wants to achieve. Based on the state of the environment in a specific time (stored in the Beliefs and in some agent local variable), the agent choose the best intention to accomplish for that specific time. Intentions are future directed and trigger certain actions [**?**]. This framework facilitates flexibility in generating actions, because an agent can switch between intentions depending on the state of the environment.

The bus interests are shared between the fleet: every bus wants to minimize the costs, the average travelling time of the passengers and the number of messages sent. However, there are some cases in which the agents compete between each others to obtain a resource, as we will describe later.

The cooperation and competition behaviour are generally achieved through communication: different agents can send messages to each others. A message is formalized through the use of an ontology, for which the string of the message is splitted in a *message type* and in a *message content*, as follow:

```
"message_type message_content"
```

For example, it is possible to ask to a bus to drive according to a specific schedule by sending it the following message:

```
"schedule 2"
```

where 2 is the ID of the schedule.

3 Conceptual description and strategy

This section explains the various agent concepts used, how they are designed and implemented, and along which strategies. Four types of costs shape our strategies, the amount of money spend, the average waiting time, the amount of messages send, and the amount of people waiting. Costs arise by traveling and buying more buses. To reduce costs, the buses need to transport travelers efficiently. The more passengers they can transport in the shortest amount of time, the lower the costs will be.

3.1 Transportation and Coordination

Buses are allowed to travel according to a specific route. This route describes the connections between different bus stops. In the following paragraphs, we will focus on how we chose the different schedules and how the buses are traveling through them.

Different fixed schedules In order to reduce the amount of time a passenger spends in a bus, the map is split into four different areas. They can be classified under the labels "West", "North", "Center" and "East". Thus, each area defines the set of bus stops composing a schedule.

Splitting the bus stops has two advantageous effects. A bus focuses only in its specific area: thus, it has to complete a shorter distance before passing the same stop again, so waiting time will be reduced. The time that a traveler spends in the bus will be shorter, which will reduce chances of buses being full and unable to pick up other travelers.

This solution does occasionally require transferring passengers from one area to another. This reduces the advantageous effects. Transfer passengers will have to wait at two bus stops before reaching their destination. To enable transportation of people between bus stops in different schedules, we defined the concept of *joint position*. A *joint position* is a bus stop that is shared between different areas. Thus, if *bus stop 4* is shared between *area 1* and *area 2*, we say that *bus stop 4* is a *joint position* between *area 1* and *2*. Stop 3 (*Centraal*), is a shared stop for all four areas. As a result, when a passenger that needs to change area in order to reach his destination is picked up, the bus will consider as the passenger destination the nearest *joint position* with the desired area: the agent will bring the passenger there and it will drop it; then, an agent from the destination area will picked up the passenger from the *joint position* and it will bring the passenger to his real destination. In order to avoid useless movements of passenger, we define some conditions to pick up a passenger from a bus stop:

- the passenger destination is in the bus area, **or**

- the passenger destination is not in the bus area **and** the passenger is not already in a *joint position* with the destination area.

Direction-based pick up Within an area, we define two different schedules. The first one is a route through all the bus stops of the area, the second one is the reverse of the first one. Thus, we can have bus traveling in the same area but following two different directions. Thus, we improved the traveling time by moving passengers along the direction that is faster in order to reach the destination. This adds a new condition to check whether a bus should pick from a stop to the ones listed in the previous paragraph: the passenger has to be picked up only if it is faster to move the passenger to the destination (or to the *joint position*) using the bus schedule, compared to the opposite one. If this condition does not hold, the passenger will not be picked up, and he will wait for another bus that is traveling with the opposite schedule. This increases transportation efficiency, since buses carry passengers for a shorter amount of time and are thus able to carry more passengers.

The number of buses assigned to areas and schedules is distributed based on the needs of that specific schedule, as we will explain later. In this way, schedule with higher need of buses will be provided with new buses.

3.2 Roles

To ensure that the buses cooperate efficiently, they are assigned specific roles. Roles differ in their ancillary responsibilities and so a hierarchy of buses was created.

Roles can also define the *behaviour* of a bus. Using roles makes it easy to change the behaviour of a bus.

Global coordinator A major responsibility is buying new buses, in case of shortage. If all buses would be able to, this would happen inefficiently and increase costs unnecessarily. Therefore only the first created bus with `ID = 24` is assigned this responsibility. This bus is on top of the hierarchy, and its role is defined as *global coordinator*.

The *global coordinator* adds buses based on shortage. Shortage is defined as the difference between the total number of people waiting, and the total bus fleet capacity. The cheapest bus type that is able to manage the amount of waiting people, is added. Thus, when a shortage is experienced, the global coordinator will create new buses: as we will explain in the following, the newly created buses will be items of auctions in order to assign the bus to a specific schedule.

Local coordinators One step down in the hierarchy, are the *local coordinators*. They are each responsible for one of the four areas in Amsterdam. Note that the *global coordinator* is also a **local coordinator**. The *global coordinator* is assigned to the "Centre", whereas the buses with `ID = 25, 26 and 27` are responsible for "West", "East" and "North" respectively. The local coordinators keep track of their area, by listing the buses assigned to their area, their fleet capacity and the number of people waiting. When a new bus is created, all the local coordinators compete between each other to assign the new bus in their area, as described in Section 3.5.

Scouts Less important than the local coordinators are the *scouts*. Note that the global coordinator and the local coordinators are also scout. The scouts are created at the beginning of the simulation: each schedule will have one scout. Their aim is to ensure that a bus in a specific schedule always exists. They do not have specific responsibility, but the semantic of the role specifies that they can not be moved from their schedule.

Simple Finally, at the bottom of the hierarchy we have the simple buses. Every bus is also a simple bus. However, the bus created after the beginning of the simulation are **only** simple. This means that they do not have specific things to do, but they only have to pick, move and drop passengers.

3.3 Communication

As stated before, to enable social interaction between agents with conflicting or agreeable goals, a form of communication is implemented. This communication happens through the exchange of messages. A bus sends a message with a specific goal, and the receiver performs actions based on the content.

Ontology To take into account the possibility of messages with different purpose, we defined the following ontology for communication: `"message_type content"`. Where `message_type` is the header of the message, or the performative [**?**]. This is a string that describes the goal of the message. Then `content` contains the effective content of the message. We implemented this ontology, so that buses know how to use the content of the message for, based on the performative. The perfomatives of the messages can be promoting, assigning, requesting help, offering help, accepting help, requesting a bid, bidding and reallocating.

The reception of the messages goes as following. At every tick a bus checks its incoming messages. It can retrieve the new messages from the inbox history. To check whether a message is new, its tick is compared to the last tick at which the agent checked its inbox. This last tick is represented by a *locally stored variable*. The performative of the message, implies how to interpret the content. Based on the content the bus could changes its intentions: for example, when a local coordinator gets a message of type "*bid-request*", the new intention of performing the bid will be added.

3.4 Group decisions

The division of the route into four areas can lead to certain scenarios that involve unnecessary costs. Steps are taken to deal with a situation, where one area has a shortage of buses whereas the other area has a remnant.

Whenever a bus is created and allocated to a new schedule, it send a message to the corresponding local coordinator, in order to make it keep track of all buses that are working in that area. In addition, local coordinators compare the amount of people waiting in their area with the fleet capacity. Based on this comparison, they decide whether they need more buses. If this is the case, they collaborate with the other coordinators.

This collaboration is done by sending a message to the other coordinators. The message has performative `help-request` and the content is a number,

which represents the difference between the local fleet capacity and the amount of people waiting in that area. The other coordinators ask to their fleets if there are empty bus which can be transferred (i.e., performative `empty-check`), and if so the empty buses send a message with performative `reallocable` and content the id of the bus. Upon receiving this offer, the requester evaluate how many buses it needs to compensate the difference between the fleet capacity and the amount of people waiting, adding these buses to its fleet and sending them the message `reallocated` with content corresponding to the local coordinator id. The other coordinator removes the bus from its bus fleet when it receives a message with performative `bye-bye`.

3.5 Negotiation

The local coordinators compete between each others in order to obtain the newly created buses for their area.

When the global coordinator creates a new bus, it also sends a message `bid-request` to every local coordinator, that will start the auction for the new bus. Every local coordinator will then send a message `bid` to the local coordinator specifying the need of the two schedules in its area. The need for a specific schedule is computed counting the number of passengers in the area that should be picked up by a bus following the schedule (remember that based on the schedule, the direction of the bus changes, and passengers will be picked up based on the traveling direction). After receiving the bids from the local coordinators, the global coordinator will choose the schedule for which the need is higher, and it will specify to the newly created bus that it should drive using that schedule, sending it a message of type `schedule` containing the ID of the schedule.

3.6 Stopping buses

We allow a bus to stop when some conditions are met. In this way, we can reduce the traveling costs when possible. In particular, a bus can stop when:

- the bus is empty **and**
- there are no passengers waiting in its schedule; here, we consider the number of passengers waiting in the bus stops composing the schedule of the bus that have to be picked up (considering the direction of the bus).

However, as we will describe in Section **??**, not allowing the bus to stop brings to better results. Thus, we deactivated this feature.

4 Code

For clarity and readability, the code is divided in different files: (i) the file `agent.nls` implement the decision process of one agent: the management of beliefs and intentions; (ii) the file `init.nls` is used to initialized all the local variables of the agents; (iii) the file `execute-intentions.nls` implement the actions related to every intention; (iv) the file `messages.nls` implements some functions to interact with a message, for example the retrieval of the content of

the message; (v) the file `utils.nls` implements all the utility functions used by the other files.

5 Tuning

The agent logic consists of a set of parameters that have been tuned for performance purposes. We tried different combinations of these parameters in the training set and in the two test sets, in order to find the best parameter setting. In particular, these parameters are:

- **stop**: whether we allow the bus to stop or not, as explained in Section 3.6; the tuning showed better results when we do not allow the buses to stop
- **initial bus type**: the type of the buses created at the beginning of the simulation (the scout buses); we set this parameter to 3, i.e., the firstly created buses are red
- **threshold for creating a new bus**: as we described in Section 3.2, the global coordinator creates a new bus based on the shortage, i.e., the difference between the number of people waiting and the fleet capacity. The shortage is compared with this threshold: when the shortage is higher, new buses will be created; we set this parameter to 0, i.e., as soon as we have a shortage, we need to create new buses
- **threshold for the bus types**: when the shortage is higher than the previous threshold, new buses will be created. The type of the newly created buses is determined by how much high is the shortage. Thus, we have three thresholds for the three bus types respectively; we set the threshold for creating a red bus to 60, the threshold for creating a yellow bus to 12 and the threshold for creating a blue bus to 0. This means that we create the cheapest bus that can "manage" the shortage situation.

6 Performance

To establish the performance of the final code, a comparison is made to a baseline code. In the baseline code, buses only travel, pick up and drop off passengers, and create new buses. Table 6 shows that there is a significant decrease in the travel time. In the final code, more buses are added which explains the increase in bus expenses. Also, the number of messages is increased, to enable cooperation between buses.

From figure 3 can be gathered that there is a peek in the number of passengers waiting during the morning rush hours. This peek lowers, when more buses are added. Thanks to the enlarged bus fleet, a smaller peek is seen during the second rush hour in the evening.

7 Performance on test data

Table 2 shows.

Measurement	Baseline	Final Code
Avg travel time	150	72.53
Buses' expenses	1039682	711251
Messages sent	0	285
Final amount waiting	232	255
Avg travel time remaining	100	78.12
Final avg travel time	156	75.48

Table 1: Performance results

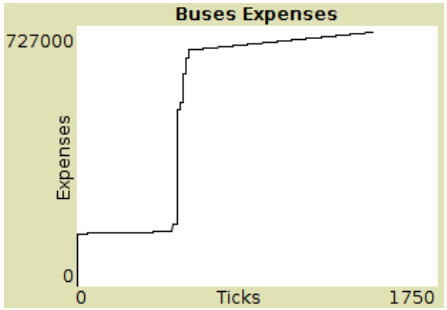

Figure 1: Figure that displays the expenses of the buses.

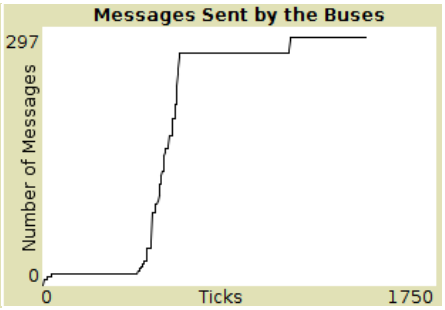

Figure 2: Figure that displays the number of exchanged messages.

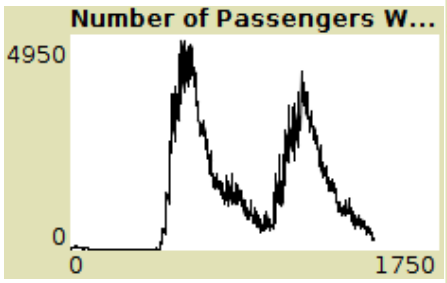

Figure 3: Figure that displays the number of passengers waiting.

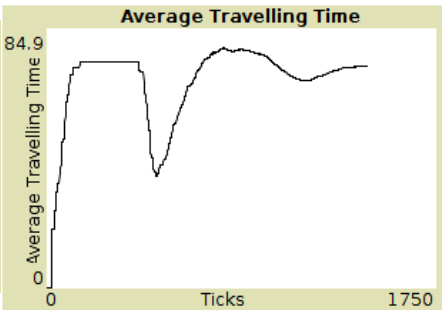

Figure 4: Figure that displays the average travelling time.

Measurement	Test Data set 2	Test Data set 3
Avg travel time	0	0
Buses' expenses	0	0
Messages sent	0	0
Final amount waiting	0	0
Avg travel time remaining	0	0
Final avg travel time	0	0

Table 2: Performance results

7.1 Conclusion

From the results can be concluded that the number of passengers and the waiting time significantly decreased for each data set that is tested. The costs and the amount of messages did increase.

8 Discussion

Some implementations that could have led to better performance of the system are the following.

The amount of occupied spots in each bus is local information, that can only be made known to other buses through communication. This information could have been used to make a more accurate estimation of the amount of buses needed for the travelers. Since the current system is very sensitive to adding new buses, the information did not seem valuable. To reduce the amount of messages send, this information is not communicated.