UNIVERSITY OF AMSTERDAM

Final Report

Luca Simonetto - 11413522
Nedko Savov - 11404345
Fabrizio Ambrogi - 11403640
Multi Agent Systems

October 18, 2017

1 Introduction

The following report analyzes and discusses every aspect of our work, and gives explanations for each decision made at every landmark of the development.

Our system can be formally defined as a multi agent system of simulated buses in a city environment, where they cooperate with each other in order to maximize a multi objective function, resulting in a Pareto of solutions that have been analyzed in order to retrieve the combination that was best suited to solve the given problem. Each bus can be seen as an hybrid[3] data-driven agent, where aspects of its learning behavior are influenced by the data initially provided. Cooperation in the system is in the form of multiple coalitions [6], whose members bargain in an alternated offer model fashion[10] in order to acquire access to the scarce resource of the system, under the form of new buses to buy.

The performance of the system is determine by the results produced in three different categories: average passenger travel time, total number of messages sent and total money spent.

2 Implementation

This section gives a brief explanation regarding the NetLogo implementation of the project, giving details on the overall functions that have been created in order to understand the basic workflow of the simulation. As every aspect of the multi agent system is explained in more depth in the following sections, only the overall functioning is described.

Bus variables

As each bus beliefs cover only a local view of the system, we added a number of variables that are required to make autonomous decisions when needed and keep track of the environment, as well as some support variables, holding details that are independent of the actions of other buses. As the environment is well defined and its basic structure is dynamic [1] (except for stations, that are kept in place for the whole time the simulation runs), we stored in each bus the necessary data that allows them to compute the shortest distances between every pair of stations, in order to always take the best path when traveling to a destination. These distances have been calculated with the help of a Python script using Dijkstra's algorithm, in order to reduce the effective code used and improve the provided code's readability. An important aspect of the project that caught our attention was the usage of real data regarding the distribution of passengers for each station through time: as the fluctuations in passengers between days are not very high (in the real world), apart from working/vacation days or sudden worsening in weather conditions, we decided to manipulate the data and add it as stored information for every bus. This knowledge would then be used in order to predict future fluctuations of passengers, allowing the agents to be more reactive when deciding which station to move next.

Bus initialization

When a new bus is created, it gets initialized with the above mentioned variables along with a mother station, that indicates which will be first station that the bus will have to travel to: this ensures even distribution of the overall agents in the city, in order to be more reactive when passengers start to populate the stations. The bus also receives an affiliation to one of two factions, *fast buses* and *cheap buses*, that will be used when making group decisions and voting.

After the initialization has finished, the bus travels from Centraal station to its mother station and starts the main routine.

Initial system state

As soon as the simulation starts, one bus is bought for each station, in order to get full coverage of the city. As the simulation starts at midnight, the number of passengers to be transported is very low and consequently the buses are mostly idle. In order to maintain a fast response when early morning passengers will enter the system, each bus tries to move to an empty station as soon as it detects that there are no more passengers to transport. In this phase no new buses are ordered, and is mostly considered a setup phase.

Actions execution

At each timestep of the simulation, every bus takes a number of actions in a fixed order, going from the handling and analysis of the incoming messages to the decision of the next station to move to.

- **Messages handling** First, the inbox is analyzed and all the messages are read in order: in the case of informative messages the current variables are updated, whereas in the case requests a corresponding action is taken.
- **Voting handling** If a voting session has started in order to decide if a new bus is required, the bus answers by giving its preference, mainly biased by the objectives of the coalition in which the bus is into. Buses affiliated with the cheap coalition will often vote against a new purchase, as their objective is to have as few expenses as possible, whereas buses affiliated with the fast coalition will probably vote in favor of the new purchase, as the increase in transportation efficiency will probably increase.
- **Beliefs update** In this step, each agent analyzes the status of the environment in order to update its current beliefs on the position of the other agents, by checking the distribution of passengers across the various stations.
- **Passengers dropdown and pickup** Completed the three above tasks, every bus drops at the current station all of its passengers. This action allows the bus to pick up only the passengers that will be worth moving to the next station as the total distance to their destination will be decreased. The decision of where to move next will be based on the number of passengers that are currently waiting at the current station, their destinations and the capacity of the bus.

3 Approach

If we consider the environment of our problem at a specific time, we see we can extract enough data to make good reactive decisions[2]. Our multi-agent system heavily relies on decisions like that. However, performance can be improved by also keeping beliefs of the state of other agents and using predefined data about the environment - in our case, the expected bus stop load throughout the day. Even though we are not using ontologies of rules, this data still defines a deliberative part of our multi-agent system.

Figure 1 shows the general structure of the agents:

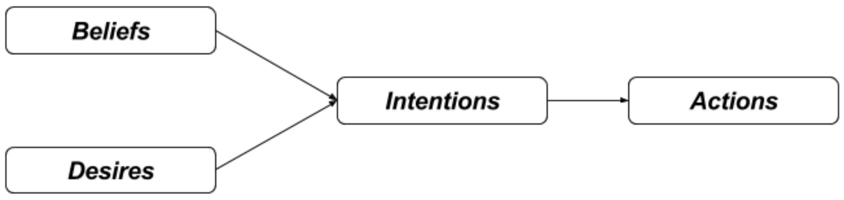

Figure 1: reasoning model for the agents.

the reasoning system follows a belief-desire-intention model[8], where *beliefs* are composed by the internal variables of the agent such as passenger predictions, bus locations etc. The common desire for every agent is to transport every passenger to their destination, and specific desires are coalition-dependent, like being efficient and quick (fast buses) and save as much as possible (cheap buses). Intentions arise from the specific states in which an agent can be, ranging from simply traveling to the next station, to drop passengers, help other agents or ask for help.

3.1 Self-gathered Information

Our system has been designed with the objective of making every agent able to determine their future intentions with the least amount of interaction possible with other agents, in order to limit the number of messages sent and emphasize the locality of information. As the majority of the decision-taking process is located at the pickup-dropdown phase, we tried to reduce the inter-agent exchange of information to the minimum. Each agent decides the next station to head to based on multiple factors, all of which can be gathered solely from the current station, its connected stations and the passengers waiting in those stations.

First, the best station to move next is determined, as a recursive calculation of the attractiveness of the surrounding stations connected to the current one. Attractiveness is defined as:

$$attr(station) = \frac{waiting\_passengers}{2} + \frac{prior}{2} - 50 * bus\_count\_for\_stop$$

where *waiting_passengers* is the number of passengers waiting at that station, *prior* is the predicted number of passengers that will arrive at that station at the next 15 minutes mark, *bus_count_for_stop* is the number of buses at that station (determined by the analysis of the agent's beliefs of the buses).

Having defined attractiveness for a single station, the recursive attractiveness for a station is given by:

$$rec\_attr(station, depth) = attr(station) + \sum_{s \in CS} \left(\frac{rec\_attr(s, depth - 1)}{ln(distance(station, s))} \right)$$

where *depth* is the recursion level (starting at 2), *CS* are the stations connected to *station*, *distance(station, s)* is the Dijkstra distance between the stations.

Having determined the recursive attractiveness of each connected station, the agent combines the results with the gain that each passenger would have, with gain defined as the amount of distance traveled towards its destination.

$$gain(passenger, s, new\_s) = 1\ if\ Dijkstra(p, s) > Dijkstra(p, new\_s);\quad 0\ else$$

In this formula, *Dijkstra* is the computed Dijkstra value between the destination of passenger p and the station s. As with the previous formula, this value is stored in the agent's beliefs, as explained in section 2.

The intention of the agent is defined as moving towards the station that maximizes the total gain from the decision, formally:

$$next\_station = \arg\max_{s \in CS} \sum_{p \in P} 2 * gain(p, current\_s, s) + rec\_attr(s, 2)$$

Having determined its intentions, the agent picks up every passenger that has a gain and travels to the chosen station.

3.2 Communication

For communication between buses we use the provided way of sending messages. Messages can be sent at each tick after an agent analyses its position, performance and the surrounding environment, they can be received in the same tick or the next one depending of the order in which agents are analyzed. If an agent sends a reply, it is expected to fully deliver it within a specific time delay, depending on the type of message. Handling of messages is the first task an agent does at each tick.

Frameworks like KQML[5] can be suitable for generalizing handling messages of a certain category (categories are related to speech acts), as many different types of messages can be sent using the same form by just changing the content. As the exchange of messages between agents is rather small, we implemented a custom structure, in order to avoid complex frameworks that would overcomplicate a rather simple task, along with a more verbose code. Messages have a variable number of custom parameters, but can still be related with different kind of speech acts, in order to ensure compatibility with a possible future communication framework. Each message has the following internal structure:

$$message = (message\_tick, sender, message\_content)$$

$$message\_content = (message\_type, parameter\_1, parameter\_2, ...)$$

- **message_tick** contains the tick in which the message has been sent, allowing to get an exact ordering of the overall messages and compare it with a threshold in order to check if it's still valid.
- **sender** id of the sender of the message, in our case the bus_id.
- **message_type** a string indicating the type of the received message, one of the following:
  - **init** this message is sent to every bus that has been created from a votation, as soon as it enters the system, in order to set some important parameters, like the mother station, the number of existing buses in the system and the initial bus beliefs, which are the same of the sender.
  - **new-bus** sent to inform the other buses that there is a new ordered bus available, so their beliefs can be updated. The message contains the new bus destination, and is mainly used to understand how many buses are in the system, toi avoid overcrowiding.

  - **order** request for ordering a bus of a selected type and for a selected station. Receiver always fulfills the request. It is sent to a "leader" between the buses, which accomplishes more the role of a burocrat, sending the init message to the rigth buses so that there are no one without initialization or with theirs overwritten.
  - **ask-vote** initiates voting. The precondition of this message is based on the expected passenger load of the station and the current bus load. The sender thinks that a new bus is needed to better serve thje station, but before ordering it it needs the permission of its close peers.
  - **vote** indicates that one bus has sent its vote in answer to a specific *ask-vote* message. It can be positive or negative, and in some cases it will bring a proposition to reach the station, to avoid buying a new bus.
  - **confirm-help** sent when accepting a support offer from a votation, the receiver heads to the station where it is needed.

Each of the messages can be assigned to a speech act category[9].

- **init** is declarative
- **ask-vote** is directive
- **new-bus** is representative
- **vote** is expressive and can be commissive (whether the bus propose its support)
- **confirm-help** is directive

3.3 Competition

Factions in this system have been added in response to the multi objective nature of the problem: as the final performance is evaluated using three different criteria, we felt necessary to implement a way in which groups of agents work together to optimize one specific objective.

As messages sent are mostly unrelated to the behavior of the buses, we concentrated on the remaining two aspects to optimize, average traveling time and money spent. All buses in our system are assigned upon purchase to one of two disjoint factions, *fast buses* and *cheap buses*, and their behavior emerges when deciding to buy a new bus: a member of the first faction will mostly vote in favor of the purchase in order to decrease waiting time thanks to the new addition, while a member of the second faction will often vote against the purchase in order to save money.

We worked with two different implementations of faction assignment: The deterministic assignment of buses, based on the bus_id, keeps a high level of reproducibility of the results and allowed further improvements to be better analyzed. This has been used for most of the "optimization" phase. The casual assignment is, in our mind, more realistic. We obtained results with a greater variance with this implementation, but never a drastic loss or gain in performance and always around the same expected values.

We made the decision to not have two coalitions of the same size, instead we opted for a 2 to 1 ratio between *fast buses* and *cheap buses*, as in the evaluation phase we know that average travel time will be twice as important as money spent. Further tuning of this ratio led to decreases in performance, indicating that 2 to 1 is a reasonable value for the two coalitions sizes.

3.4 Cooperation

In order to keep the system as efficient as possible, multiple forms of coordination have been implemented, allowing the buses to know each others intentions and act accordingly.

In our agent system the desire of every agent is to bring as much passengers as possible closer to their destination. keeping that in mind, they make a decision at every station where to go next. However, if this decision is not brought in line with the intentions and decisions of the other agents, many of them might end up at the same station where they want to pick the same passengers, which will have a negative effect on the performance of the whole system. Therefore, a coordination technique is needed.

The decision of where to go next for an agent can be thought of as a partial plan of the agent[7]. In our case, an agent keeps track of its beliefs the presumed position of every other agent and if it detects that there are too many other agents directed to a specific station, the probability of moving there will be greatly reduced. This approach determines the multi-agent planning component of the system, where agents use local plans to achieve a global objective.

The beliefs regarding other buses destination is made through observation of the environment. At every tick the agent compares the state of the world with the one of the previous tick, more specifically it observes which of the passengers that where waiting in a station are not there anymore. With this information the bus understands that another agent has picked them up and, based on the protocol of minimizing the distance of the passengers from their final destination, it can infer where the bus is headed with those specific passengers. It is important to note that this information is an estimation, as one of the objectives was to avoid keeping complete information of the world. Although some particular passenger fluctuations in the system could lead to miscalculations in the agents beliefs, they are unlikely to overcome the estimations in the long run (since a different destination would have meant a different passengers collection most of the time).

Another form of cooperation that our buses perform is waiting: if a bus is stopped at a station and detects that some passengers are being transported to the same station, it can decide to wait for the second bus to arrive. This would allow both buses to exchange the respective passengers in order to maximize their efficiency, by moving more passengers closer to their destinations. Coalition affiliation has an impact on the approach taken to the waiting protocol: while cheap buses tend to wait often in order to save money (increased by traveling) and time, fast buses generally are more impatient, resulting in less occasions where waiting is considered a viable option.

The waiting choice protocol is a bit complicated, but it's core is based on time person-efficiency. If the bus waits in the station the passengers it should have moved will wait as well, so we have a loss, the protocol tries to predict if from the passengers coming to the station a bigger gain is achieved by having the two buses leave the same tick. The passengers brougth by the next bus may actually be loaded by the waiting one for the same direction, so that they will not need to wait for another bus reaching the station.

3.5 Group Decisions

When a bus has reached its maximum capacity and is forced to to leave some passengers at a station, it checks if having a new bus in the area would bring enough benefits to motivate a new purchase. This protocol takes in consideration how many passengers are left down, how many buses are headed to that station and how many buses are operating in the whole system. If it determines that a new bus is needed then a new voting session is started, by sending an *ask-vote* message to every other bus. As with waiting, coalitions play an important role in this task, as fast buses are more inclined to start a voting than cheap buses: the probability by which a bus decides to request a bus is controlled by a faction-dependent threshold.

When voting[4] for a new bus, we apply a plurality vote where only the nearest buses (within a radius of 10 steps surrounding the station) are asked to act. To do that, the buses' actions are pruned by a simple rule-based social norm. It will not allow a distant bus to vote. The voting procedure counts equally every bus contribution, giving that they are sufficiently close to the station in which the requesting bus is located. Agents that belong to the cheap faction will be more conservative on the amount of buses around, and will vote against a new one more often than not. This brings us to the implementation of a surrogated competitive intelligence, which was set by design.

3.6 Negotiation

As stated in section 3, the *fast* coalition is twice as big than the *cheap* one. This means that is likely that voting for a new bus will result in a positive outcome, as most of the fast buses will vote in favor. In the case a cheap bus is asked to vote, it can decide to stop the current transport schedule and move to the station of the requesting bus: in this way the issue can be quickly solved by transporting the passengers that pose a problem, along with ensuring that its desire of saving money is met. This action puts social utility in front of the individual agent's, and reduces the overall system commitment to the purchase of a new bus. Since this outcome would not change the result for a fast bus, no opposition is made.

In our implementation fast buses are straight-minded while cheap buses are strategic, because they need to overcome their minority. With this negotiation they have to sacrifice their personal performance to minimize losses in their utility.

They do that by trying to act like a fast bus - they run their decision function with the parameters of a fast bus. If the result of that is that a fast bus would vote yes, the cheap bus now considers if it can go help. If it does that and the voting organizing bus agrees, it will go and in this way will prevent buying of a new bus. The bus responsible for the voting gets the deal for help and decides if to accept help or order a new bus. The decision is based on the distance to Centraal and the distance to the helper. As it can be seen, this is a one round negotiation.

In our implementation we give chance to more buses to offer help by assigning a separate radius around the voting station in which buses can help. This means if there is a larger radius the voting is not affected but still buses from far away can send help messages. However, only if the bus responsible for the voting is a cheap one, it will consider help messages from the full radius. A fast bus will tend to ignore help offers from far away as to prefer ordering new buses. However, when experimenting, we found that a smaller radius works best, so we fixed this radius to the one for voting, which is of value of 10 steps.

4 Results

The system with the final implementation of all of the components described above gives the following results:

	Initial type	Ordered type	Time	Money	Messages
	Red	Red	47.63	722 558	1827
	Yellow	Red	48.25	726 592k	2326
	Red	Yellow	46.5	642 592	2210
	Yellow	Yellow	48.93	593 556k	2898

Table 1. Performance metrics of the system for each type of bus sizes combinations. Initial type is the bus type of the number of buses created at the beginning of the simulation, ordered type is the type of the subsequently created buses.

We chose not to use small size buses (green) because they prove to be too small and inefficient to use as a basic type of bus. They can help as separate individuals and support for other major type of bus but this was not implemented in this system.

It can be seen that the best average time we get from large size initial buses and middle ordered buses. This is also the most balanced result between time and money that we had. The reason is possibly because at first in the morning a lot of buses are needed at once. Since ordering new buses is governed by voting procedures which require some initiative from the buses and time many buses will not be ordered at once. In this period the starting big buses manage to cope with the large demand. Then smaller buses are ordered which seem to be more suitable in the long term - they don't spend so much money moving around and if they are too full new buses will be ordered meanwhile. Big buses will still help a lot in these situations.

The best money we achieved with all medium sized buses. This also means this combination of buses is very conservative with money usage. The reason is that the rules in the system are mostly tested on the red-red bus combination (since it performs worst, we try to achieve best performance everywhere by testing on it). The rules obtained for it, however, don't guarantee best balance for the rest of the combinations. Since yellow buses inherently spend less for moving around in the city, they can generally afford to spend more for new buses. A rule system for achieving this, however, was not implemented.

The worst result we got for middle sized initial buses and large ordered types. The red buses positively affect the result, as discussed before. This claim was also confirmed by the fact the the tested combination of all big size buses (red-red) performed better than the one discussed currently (yellow-red). The large buses also influences the outcome negatively because they spend more by simply moving around the city and a lot of their capacity will be unused during non-peak hours.

5 Conclusions

This implementation showed that a multi agent approach can be used to solve a real world problem of prime interest for modern societies such as mass public transport. The proposed approach tackles multiple aspects of this problem like partial planning and cooperation, exposing solutions that are supported by the experimentation process. Considering that the strategies taken used general principles and techniques, this project could be adapted to a variety of similar situations posing similar issues and challenges without putting much effort into adapting the whole code. The use of existing data for passengers prediction indicates the possibility of integrating more sophisticated prediction techniques such as neural network and tools alike.

6 Improvements and future work

6.1 Communication framework

A welcome addition to this project would be the usage of a full-featured framework, in order to allow for a greater number of message types with a well defined structure. The KQML or FIPA frameworks seems to be suitable enough, especially the lattermost, as it has a fewer number of different preformatives. In the current stage of development no other message types are needed, but with the additions of more features, a professionally developed framework would reduce unnecessary problems.

6.2 Dynamic usage of bus types

Currently, only one bus type can be chosen for the consequently created (ordered) buses. This could be changed by also including in the voting a way to vote for bus types and order the winning type. This could lead to creating buses with appropriate sizes chosen for the specific needs.

6.3 Passengers prediction

This project has a predictive component under the form of passengers prediction: given the next 15 minutes time slot, every agent is able to predict how many passengers will show at every station. At the moment this predictions are simply a copy-paste of the schedule given with the project files, but with the addition of more data a complete prediction mechanism could be implemented in order to catch fluctuations between days and time slots.

References

[1] Multi Agent Systems, Section 2.1, *Environments* pag. 17

[2] Multi Agent Systems, Section 5.1, *Brooks and the Subsumption Architecture* pag. 90

[3] Multi Agent Systems, Section 5.3, *Hybrid Agents* pag. 97

[4] Multi Agent Systems, Section 7.3, *Negotiation* pag. 137

[5] Multi Agent Systems, Section 8.2, *Agent Communication Languages* pag. 168

[6] Multi Agent Systems, Section 9.1, *Cooperative Distributed Problem Solving* pag. 190

[7] Multi Agent Systems, Section 9.6, *Coordination* pag. 200

[8] Slides of Lecture 1, *Multi-Agent Systems*

[9] Slides of Lecture 4, *Understanding Each Other & Communicating*

[10] Slides of Lecture 11, *Bargaining*