A Multi Agent System for Autonomous Public Transport in Amsterdam

Juan Buhagiar
University of Amsterdam

Sarantos Tzortzis
University of Amsterdam

Yiangos Georgiou
University of Amsterdam

https://github.com/sarantinio/Multi-Agent-System-.git

March 28th, 2018

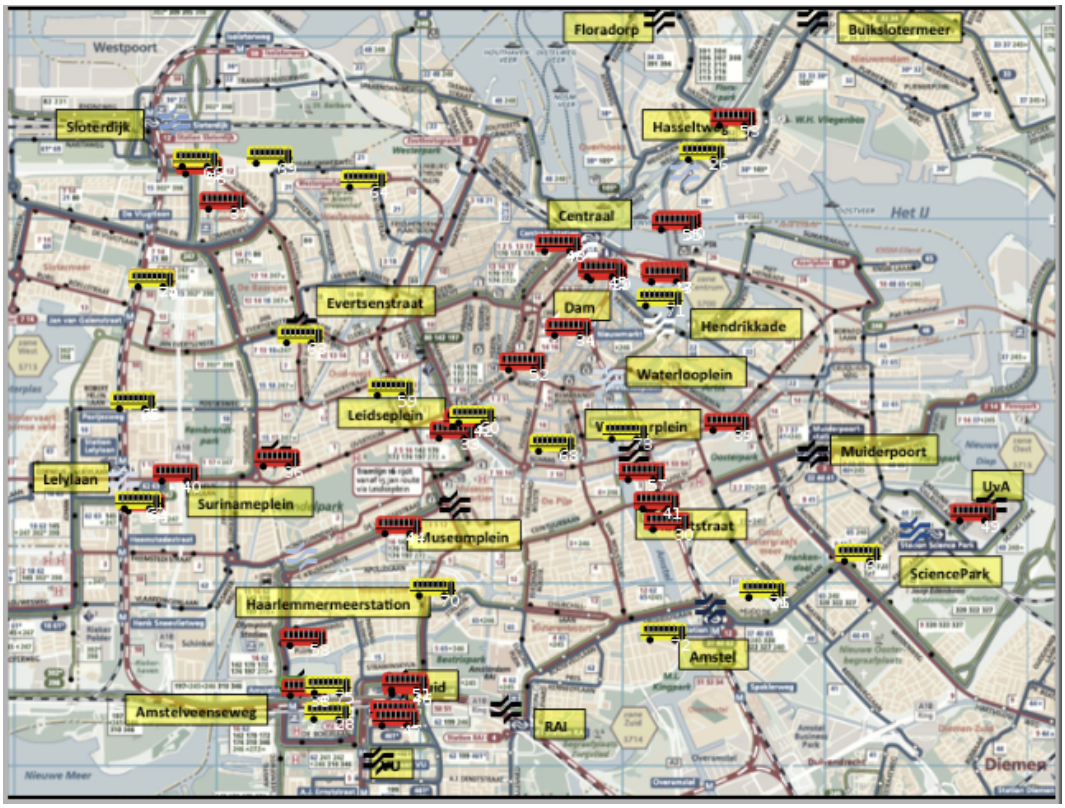

Figure 1. Net Logo Simulation.

Introduction

The Multi Agent System for Autonomous Public Transport in Amsterdam is a simulated transportation scenario for the city of Amsterdam. It includes the coordination of many different intelligent agents. We have different breeds of agents such as buses, bus stops and passengers. We seek to optimize the transportation of passengers from bus-stops to their destinations with the use of intelligent buses. We will focus on minimizing the time passengers wait to reach their destination while

also minimizing the costs incurred.

Related Work

Multi-agent systems (MAS) or self-organized systems is an automated system that is composed of multiple int erecting intelligent agents. Multi-agent systems have been utilised to solve problems which are hard or even impossible with a single agent. Multi-agent systems have been proven a success in a number of fields including but not limited to unmanned air vehicles (UAVs) (Kovacina, Palmer, Yang, & Vaidyanathan,), timing (Moreau,), and search (Kitano et al.,).

Multi-agent systems in the transportation domain is a research area that has attracted attention by both industry specialist and research institutions alike (Davidsson, Henesey, Ramstedt, Törnquist, & Wernstedt,). This is mainly due to technological advancements that have enabled self-driving vehicles to be developed (Schoettle & Sivak,). Other areas of transportation that Multi-agent systems have been appllied to in the transportation domain are air traffic, rail traffic (Böcker, Lind, & Zirkler,) and road traffic (Doniec, Mandiau, Piechowiak, & Espié,).

Apart for being applied to different sub-domains in the transportation industry, MAS have been used in a variety of problems in this sector ranging from conflict management in air-traffic control (Tomlin, Pappas, & Sastry,) to fully fledged transportation management for cities (Balbo & Pinson,).

Two sub-problems that are the focus on many research papers in the MAS transportation domain are traffic management and transportation management. Traffic management is mainly the resolution of conflicts between agents that have planned a specific route. While transportation management focuses on the transportation of items, be it goods or people, from a location to a given destination.

In the paper we will focus on the transportation management problem to optimize the transportation of passengers thought the city of Amsterdam through public transport.

Methodology

The simulation provided handles the distribution of passengers across different parts of the city. Our task is to develop the coordination of the smart buses to help passengers reach their destination. We chose to implement a master/slave system such that all buses are considered as slaves apart from one agent which is the master. The buses always communicate with the master, which coordinates all the information.

The transportation simulation is designed and run in the NetLogo environment (Wilensky,).

Agent Behavior

Each bus is an agent. Each agent is an intelligent entity with a specific personal target to deliver its passengers to their destinations. In order to achieve this, communication with the other agents is needed. The agents behavior is split into two different phases. The *exploration* and *exploitation*. The agents change from phase to phase according to its current busload (*threshold*).

Exploration

When the agent has none or a limited amount of passengers boarded, his state is set to the exploration mode. Thus, it attempts to fill the free space on the bus with passengers waiting at the neighboring stations. Each bus moves to a random adjacent station and does not try to pick a final destination to deliver passengers. A restriction added here, is that an agent will want to avoid picking a station that another bus is heading there with the aid of the *Communication* protocol.

When a bus is initialized, zero passengers are on-board. The bus will determine the next stop aiming to pick the most passengers available. If the current passengers waiting on the stop are less than the capacity of the bus then all of them will be boarded. If the number of people at the bus stop is greater then the bus's capacity, the agent will choose the largest group of passengers with the same destination as the people on-board. For

the rest of the exploration state, the bus will always create a sorted list with the groups of passengers on-board grouped by destination. This list is used so that the bus can filter passengers according to this priority.

Once a threshold of passengers on-board is reached, the state of the bus is changed to *exploitation* mode where the agent determines a final destination. The state of the bus, exploration/exploitation is always communicated to the master bus (refer to the Communication protocol).

Exploitation

Once the number of passengers on-board exceed a pre-defined threshold, the agent is set to exploration mode. Once in exploration mode, the agents intention is to deliver the passengers on-board as quickly as possible.

In order to decide the next stop several factors are taken into consideration. Firstly, we look at the passengers on-board grouped by destination. Secondly, we consider the distance between the current stop and the destination of passengers. We then give a penalty to each destination if another bus has already communicated that it will go to that destination. We combine this information by using the following formula:

$term_a = (\#Passengers * 150) - distance\_d$
$term_b = traffic\_To\_Destination_d * 75$
$Weight\_d = term_a - term_b$

Where $Weight\_d$ is the weight from the current location to destination d. To calculate the $\#Passengers$ for destination d, we need to know how many passengers benefits by the path from our location to destination d, thus, we compute the shortest path between our location and d and we count the passengers who can reach their desired destination in that path. $distancec\_d$ is the distance between current stop c and destination d. $traffic\_To\_Destination_d$ is the number of buses already going to destination d. This information is local, each bus calculate the a Weight_d array with a score for each bus stop. Once we have this information, we sort the stations by this weight and select the maximum value. Once the destination of the bus is determined we find the shortest path to follow by applying the network extension (nw) by netlogo. The agent will follow this path until it reaches the last stop. At each station, it will pick up passengers if there is available space, or drop if there are passengers whose current destination is one of the intermediate stops. When the destination is met, the agent will check again its current load. If it is lower than the threshold it changes to exploration mode again, otherwise it will pick immediately the next destination applying the same procedure.

Communication Protocol

For agents to be able to communicate with each other, we designed a communication protocol such that agents are able to understand each other. Since every message sent through our communication channels costs money, we have chosen to take a master slave approach, where a master agent is selected that coordinates information between to rest of the agents/slaves.

Below we explain each *type* of message that the agents can send in order in *communicate*.

Master to Agents:

Table 1
Conflict.

'Conflict'	["bus stops with conflict"]

Conflict : In the case of which two agents are going to meet at the same bus stop during the following *ticks*, the master forward to the agents this conflict and adds to the message each agent's preference list so that they will load only passengers that fits their destination.

Table 2
Vote.

'Vote'	*Average Passengers per Bus*

Vote: When there are lot of people waiting at the bus stops, the master ask the buses to *vote* what

kind of bus should be added.A message is sent accompanied with the average passengers that each bus should load in order to zero the ones waiting.

Table 3
Default.

[4|0|3|...|2|3|0|]

(***Default***) : The master keeps updates all the agent by sending them a message informing them for traffic. The message carries a list of 24 cells. Each cell resembles a bus stop, and the value of it resembles the number of buses heading to these stops at their next station.

Agents to Master:

Table 4
Response.

'Response'	Bus type

Response : Agents send a response message. A number from 1 to 3 indicates the type of bus that an agent deems suitable to be added.

Table 5
Default 2

Bus type Bus stop Bus load
Next stop decision
["destination of each passengers"] path to the final destination

(***Default 2***): Every time an agent arrives at a bus stop, updates the agent with its current info. The message is consisted of its *bus type*,the current *bus load*, the current *bus stop*, its next *decision*(stop) and a 24-cells-list indicating the destination of each passenger that is embarked in the bus. If the bus is in *exploitation* mode the *path* of its current stop till the final destination is also added.

Cooperation-Coordination

A major target for the system is the agents ability to cooperate and have some coordination with each other. Such that agents avoid actions like going on the same bus stop. To achieve such behaviour, every time an agent is at a bus stop, it communicates its list of passengers grouped by their destination and the path of the bus to the master bus. The master bus will take the information from all buses and merge them into one list and sends this information back to the buses.This array is called traffic table and contains the number of buses that is going to each bus stop. Thus, our buses will choose the adjacent station that has less incoming buses. This will allow buses to take decisions based not only on their local information but also information from other buses destinations. Particularly, as referred to above, if a bus is during its *exploration* mode, it will try to avoid to pick a next station that another bus is already heading there.

Group Decisions

Group decisions is an important form of coordination that we have developed in the system. The simulation starts off with one small bus as time passes the demand of buses increases as passengers show up at bus-stops. The naive way of adding buses would be to add a bus every-time a bus cannot pick up passengers from the bus-stop. This is far from ideal as the addition of buses incurs costs and does not guarantee a solution. Therefore, we have decided to implement a voting system where all buses are given the opportunity to decide if a bus should be added based on local information.

The master can call a vote once it deems fit by sending a message to all other buses. The 'slave' buses respond with their vote and the master will decide weather or not to add a bus based on the majority rule.

Calling a vote. The master keeps track of the number of waiting passengers since the last vote. Before the first vote occurs, the master counts the passengers since the first tick. Once a threshold

of waiting passengers is exceeded the master will call a vote by sending a 'VOTE' message to the other buses. In the 'VOTE' message the master will include the average number of waiting passengers that each bus must pickup.

The threshold is defined by the maximum capacity of all buses. The maximum capacity is calculated by the master by taking into consideration the bus type for each bus. The bus type is communicated to the master by the slaves when they send information regarding their desired destination. The average number of passengers that each bus need to take is calculated by taking the number of waiting passengers divided by the number of buses on the road.

Voting procedure for buses. Buses will vote based on local decision. When a bus receives a 'VOTE' message, the bus will calculate how many passengers it can still carry. If this number is smaller than the average number of passengers waiting received by the master it will vote in favor of adding a new bus, otherwise, it will vote against. If a bus is in favor it will also vote for which bus type it wants to be added. This is decided based on the difference between the capacity it has available and the average number of passengers waiting.

Counting of votes. Once the master has received votes from all other buses, it will count the votes. If there is a majority in favor of adding a bus it will count which bus type is the most popular and add the number of buses of this type. If there is no majority it will not add the buses. In case of draw between two or three bus types, the bus with the biggest capacity will be added. The number of buses added depends on the number of votes for that bus type. When possible a bus type with a larger capacity is added instead of multiple buses with smaller capacity.

Competition

We decided to include competition for picking up passengers from bus stops. This will occur when buses plan to go to the same bus stop. Instead of picking up all possible passengers they will have to compete for the passengers they would like to board.

Detecting conflicts. Agents plan their path by looking at the people on board, when a decision for the next stop is taken it is sent to the master. The master will keep track next stops of the buses. The master will check if there are two or more buses that have the same next stop. If so, the master will detect the conflict. Then, the master will start a procedure in order to allocate passengers to the buses.

Passenger Preference. When the master detect the conflict it will make a competition between the buses that involves in this conflict. Master will look at the passengers waiting at the conflict stop and create a list of passenger counts per destination. Then, it will make a preference list for each bus in the conflict based on the most recent information that each bus sent to the master. The priority list for each bus follows three steps, firstly it prioritize passengers which have a destination that is in the current path of the bus. Secondly, it will prioritize the passengers with destinations matching the passengers on-board, sorted by counts. Then, we fill the rest stops.

Passenger allocation by the Master. Once the master has created all the prioritized list it must send to each bus which passengers to pick up. This is done by iterating over each element in each list concurrently. The logic behind this is to create a fair policy in assigning passengers based on the preference of each bus. For instance, if we have two preference list we take the first preference from the first list and we allocate those passengers to that bus, then we take the first preference of the second list and we allocate those passengers on this bus, then we do the same for the second preference and so forth. In that way, if bus stop s is higher in the preference list of bus i than bus j, bus i will take that group of passengers. Bus j will only take passengers that goes to bus stop s only if bus i will not be able to load the whole group of bus stop s passengers. Master also take into account the capacity of the buses and the number of passenger

that wait for each destination, in that way we can allocate precisely what group of passengers each bus that is involved in a conflict should take. Thus, a priority list for each bus is created by the master, then the master send each list to the corresponding bus.

Experiments and Results

After finishing the implementation a fine tuning was required. Experimentally we found that it is better to begin with only two buses ("*night buses*") as till the early morning only few passengers are moving. At the Figure 2 you can observe that during the first pick, passengers waiting are steeply increased but after the voting to add buses there is a balance again and so on. By the end of the fleet none passengers left that they still wait which is one of the goal of this assignement.

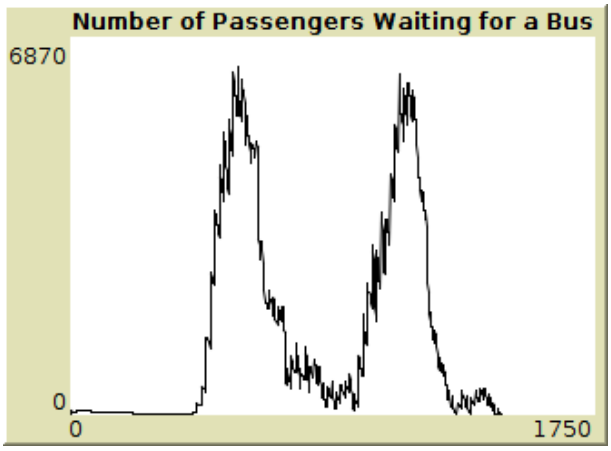

Figure 2. This plot illustrates the number of waiting passengers over a day.

Empirically, we tuned the threshold of exploration and exploitation mode and this impact can be seen in the average travelling which is gradually increased but after buses are added and/or bused are full and create path time is decreased. The illustration of this can been seen in Figure 3 below.

In Figure 4 and Figure 5 we demonstrate how the bus expenses and the message sent are escalating through the day respectively.

The exacts numbers of the simulation can be seen in Figure 6

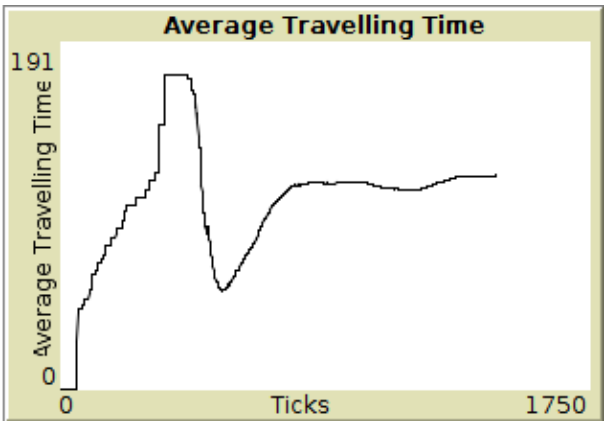

Figure 3. This plot illustrates the average traveling time of passengers over a day.

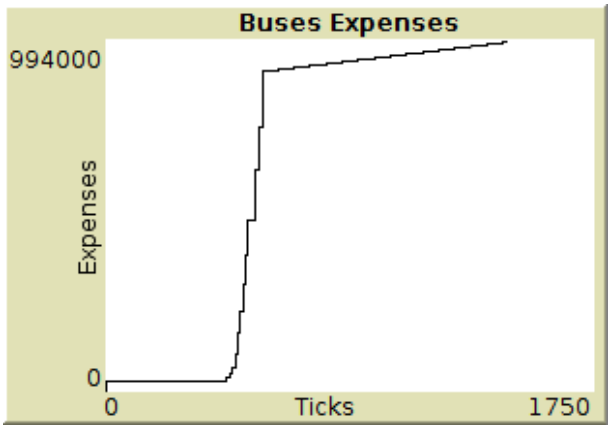

Figure 4. This plot illustrates the buses expenses over a day.

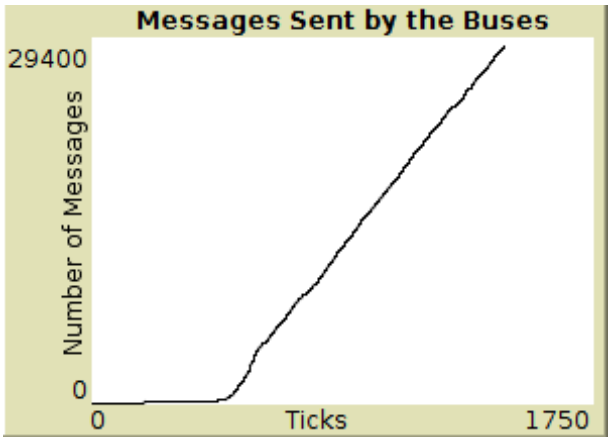

Figure 5. This plot illustrates the number of messages sent for a day.

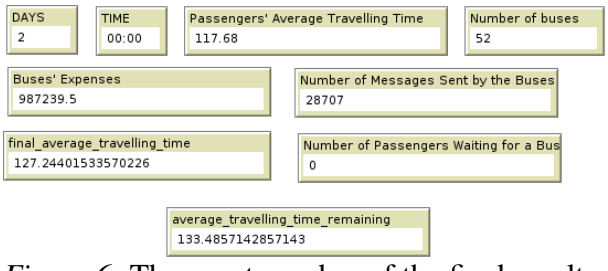

Figure 6. The exact number of the final results.

Discussion

We observed that adding too many buses the conflicts augmented a lot and the master was unable to deliver the messages in time. A more conservative add of buses appeared a fine solution. Another thing that could be added in order to reduce the average travelling time would be to split the graph in regions and the buses takes control over a specific area.

Conclusion

The project met the main goals, the agents efficiently *cooperated*, *negotiated* and *competed* in order to take decisions as intended.

References

Balbo, F., Pinson, S. (2001). Toward a multi-agent modelling approach for urban public transportation systems. In A. Omicini, P. Petta, R. Tolksdorf (Eds.), *Engineering societies in the agents world ii* (pp. 160–174). Berlin, Heidelberg: Springer Berlin Heidelberg.

Böcker, J., Lind, J., Zirkler, B. (2001). Using a multi-agent approach to optimise the train coupling and sharing system. *European Journal of Operational Research*, *131*(2), 242–252.

Davidsson, P., Henesey, L., Ramstedt, L., Törnquist, J., Wernstedt, F. (2005). An analysis of agent-based approaches to transport logistics. *Transportation Research part C: emerging technologies*, *13*(4), 255–271.

Doniec, A., Mandiau, R., Piechowiak, S., Espié, S. (2008). A behavioral multi-agent model for road traffic simulation. *Engineering Applications of Artificial Intelligence*, *21*(8), 1443–1454.

Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, man, and cybernetics, 1999. ieee smc'99 conference proceedings. 1999 ieee international conference on* (Vol. 6, pp. 739–743).

Kovacina, M. A., Palmer, D., Yang, G., Vaidyanathan, R. (2002). Multi-agent control algorithms for chemical cloud detection and mapping using unmanned air vehicles. In *Intelligent robots and systems, 2002. ieee/rsj international conference on* (Vol. 3, pp. 2782–2788).

Moreau, L. (2005). Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on automatic control*, *50*(2), 169–182.

Schoettle, B., Sivak, M. (2014). A survey of public opinion about autonomous and self-driving vehicles in the us, the uk, and australia.

Tomlin, C., Pappas, G. J., Sastry, S. (1998). Conflict resolution for air traffic management: A study in multiagent hybrid systems. *IEEE Transactions on automatic control*, *43*(4), 509–521.

Wilensky, U. (1999). *Netlogo. evanston, il: Center for connected learning and computer-based modeling, northwestern university.*