A Multi-Agent System for Autonomous Public Transport in Amsterdam

Marco Federici[*], **Dana Kianfar**[†], **Jose Gallego-Posada**[‡]

University of Amsterdam

1 Introduction

We are tasked with designing an autonomous public transportation system. Given the highly complex and decentralized nature of the problem, a multi-agent system (MAS) design is a natural approach for building an efficient solution without requiring expert engineering. The goal of our system is to minimize *1. the passengers' waiting time 2. leasing and operational costs of vehicles 3. inter-bus communication costs*. In our setup each *agent* is a bus, and its *environment* consists of other agents, passengers, and 24 bus stops represented in an undirected graph.

We investigate three approaches to designing a MAS that is an efficient and robust solution to the problem. The main behaviors of our agents are to select the travel path, pick up and drop off passengers, send messages and add new buses to the fleet. We believe that our agents' interests are to maximise the quality of the overall transportation system. As a result our agents are *benevolent* and maximizing their individual utility amounts to maximizing the overall performance of our system. Given that our agents only have incomplete information about the status and existence of other agents, we model their interactions as a Bayesian game.

Our baseline method is a set of *reactive* heuristics and decision functions, modeled in a *subsumption* architecture, that each agent uses to compute its next action given its perception of the environment and current beliefs. Selecting the travel path is the most complex behavior to model. This is due to the inaccessible and highly dynamic nature of the environment. In our experiments, we investigate alternative methods to learn this behavior using Deep Reinforcement Learning (DRL) and Evolutionary Algorithms.

We discuss our approaches in detail later, and hypothesize that our models using DRL and evolution should improve upon the baseline heuristic. Within the scope of this project, we did not observe a significant improvement of these two models over the baseline heuristic. We are are able to achieve an average passenger travel time of 71.37, a total execution cost of 1.09 M, and a total of 32.160 messages sent amongst our agents using the result of the evolutionary algorithm applied to the baseline model.

This report is structured as follows. In Section 2.1 we present an overall view of our system. Section 2.2 describes the general action pipeline of an agent. Sections 2.3 - 2.8 provide details about the models used to represent the different behaviors of the agents. In Sections 4 and 5 we present our results and conclusions.

2 Methodology

2.1 System Description

We implemented a simulator with a visual interface using `Python` as illustrated in Figure 1. Our simulator is operated by a `controller` that computes the state of the system at discrete steps which we will refer to as *ticks*. Each tick corresponds to one minute of real time, and therefore a simulated day is 1440 ticks. Each station is assigned a unique *id* ranging between $[0, 23]$ which we will use for referring to a station throughout this report. Each agent has a fixed capacity and is initially spawned at station 3. Passengers arrive every 15 ticks in a subset of station according to a dataset.

[*] `marco.federici@student.uva.nl` - Student ID: 11413042

[†] `dana.kianfar@student.uva.nl` - Student ID: 11391014

[‡] `jose.gallegoposada@student.uva.nl` - Student ID: 11390689

Following the principles of MAS, our agents do not have access to global information. As an exception however, agents have free and immediate read-access to complete information about any stations' spatial location and waiting passengers at current simulation time. No information is shared amongst agents unless they explicitly choose to communicate. Each agent sends requests to the *controller* to execute commands or access information outside its local view such as station information. At the start of the simulation, we are provided with a single bus at station 3. New buses are purchased via a voting procedure.

The environment has the following characteristics. It is *inaccessible* as no agent can obtain complete information about other agents and passengers currently en-route to a station. Furthermore, it is *non-deterministic* and *highly dynamic* as the passenger arrivals and actions of other agents change the state of the environment. Finally, the environment is *discrete* as the states are created every tick [1, Ch. 2].

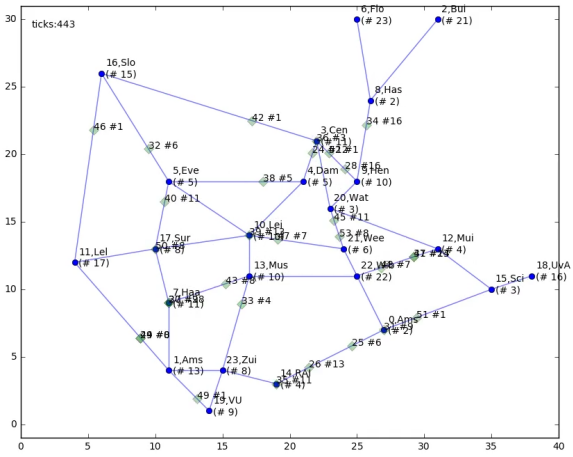


Figure 1: Example of the system simulation. Blue dots represent stations and green diamonds represent buses.

Our system consists of *hybrid* agents which have *social, reactive* and *pro-active* components. The agents asynchronously communicate to share their status and beliefs according to a policy explained in Section 2.4. Our agents are *proactive* as they select their destination according to the distribution of the cumulative waiting time of passengers over the map. Furthermore, they have *reactive* components which compute an action based on the state of the environment, which will be discussed in detail below [1, Ch. 4].

2.2 Agent Behavior

Figure 2 provides a general overview of an agent's behavior. Upon arriving at a station, all passengers which are currently in the bus are dropped off. Based on most real world transportation systems, we assume that allowing passengers to have intermediate stops during their trips could improve the overall performance of the system. The agent then perceives its environment, where it observes the current passenger distribution in the whole system. The agent updates its beliefs according to this perception and its inbox containing shared beliefs and votes received from other agents. Using its updated beliefs, the agent constructs its subjective representation of the environment, and proceeds to select its next destination towards adjacent stations. Given this decision, the agent hosts an auction among the waiting passengers at the station, who submit a sealed bid for obtaining a seat on the bus. According to a communication policy, an agent may choose to share its beliefs with fellow agents. Finally, the bus travels to its selected destination, and the full process is repeated.

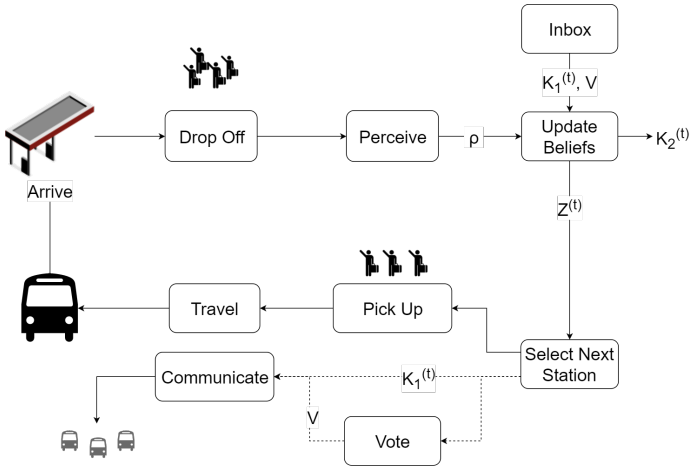


Figure 2: Flowchart of agent behavior

2.3 Notation

This section provides a formal definition of the notation and techniques used throughout the rest of the report.

We index the steps of the simulation by a time variable $t \in \mathcal{T}$. We assume that the initial simulation time is zero. We denote a station by S and the set of all stations by $\mathcal{S} = \{S_i\}_{i=0}^{23}$. $\not{S}$ represents the states outside the scope of the system. $d(S, S')$ denotes the distance between the stations, S and S'.

$\mathcal{M} \coloneqq \langle \mathcal{S}, \mathcal{A}, \mathcal{D} \rangle$ is the map of Amsterdam, where $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a binary adjacency relation between stations, and $\mathcal{D} \colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}^+ \cup \{0\}$ such that $\mathcal{D}(S, S') = \begin{cases} d(S, S') & \text{if } (S, S') \in \mathcal{A} \\ +\infty & \text{else} \end{cases}$.

A passenger is defined as $P \coloneqq \langle S_D, t_0 \rangle$, where $S_D \in \mathcal{S}$ is the destination of the passenger and t_0 is his creation time. The set of all passengers in the system is denoted by $\mathcal{P}$. Note that this set changes over time as the simulation is executed.

$S_P^{(t)} \colon \mathcal{P} \times \mathcal{T} \to \mathcal{S} \cup \{\not{S}\}$ returns the station at which passenger P is located at time t. If the passenger has already been delivered or has not been created yet, $\not{S}$ is returned.

A bus $B \coloneqq \langle C, S_C, S_N, K_1^{(t)}, K_2^{(t)}, V \rangle$, where C is the capacity, S_C and S_N are the current and next stations of the bus. $K_1^{(t)}$ and $K_2^{(t)}$ are called the *first* and *second order knowledge* databases. $\mathcal{V}$ is a set of ballots casted for the voting mechanism that decides on the purchase of new buses. The set of all buses is denoted by $\mathcal{B}$. This set is dynamic since additional buses can be added during the simulation.

The state of the bus at time t is denoted by $Z^{(t)} \coloneqq \langle K_1^{(t)}, \rho(t) \rangle$, where $\rho \colon \mathcal{T} \to 2^{\mathcal{P}}$ is the perception function shared by all the buses. $\rho(t)$ returns the passengers present in the system at a time t.

This description of the buses represents *intentional* agents whose beliefs are given by $K_1^{(t)}$, $K_2^{(t)}$, S_C, C and V. Their corresponding intentions are expressed by S_N and a voting decision. The agents are assumed to be aware of the static structure of $\mathcal{M}$ and the existance of a coordinator bus.

2.4 Communication Protocol

Communication is used among our agents to share beliefs about other agents and to submit their votes for adding other buses. These behaviors correspond to the *representative* and *commissive* speech acts [1, Ch. 7]. Formally, a message M is defined as a tuple $\langle B, B', \mathcal{C} \rangle$, where $B \in \mathcal{B}$ is the sender, $B' \in \mathcal{B}$ is the receiver and $\mathcal{C} \coloneqq \{\mathcal{V}, K_1^{(t)}\}$ is the content of the message. We will define $\mathcal{V}$ denoting a collection of votes in Section 2.5.

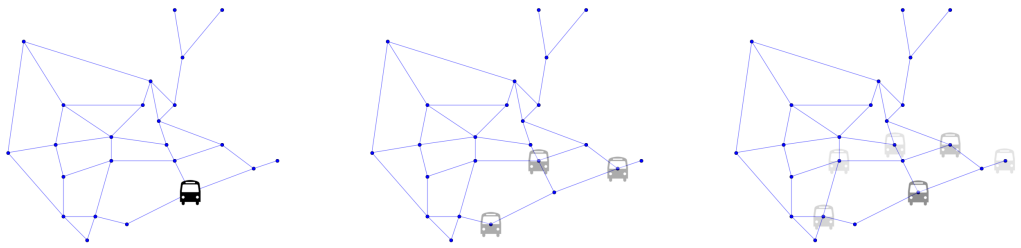
$K_1^{(t)} \coloneqq \{\langle i, S, \tau, C \rangle\}$ represents the information that B has regarding some of the fellow buses in the system. Each tuple $\langle i, S, \tau, C \rangle$ in $K_1^{(t)}$ means that B knows that the bus with id i will arrive, or has arrived, at station S at time τ, and that the capacity of this bus is C. A bus B obtains this information via messages it receives from other buses, which in turn could have received it either by indirect communication or directly from the bus with id i.

$K_2^{(t)} \coloneqq \{\langle i, \tau \rangle\}$ represents what B thinks that the other buses believe about its position. An element $\langle i, \tau \rangle$ of $K_2^{(t)}$ means that B is certain that the bus with id i was informed about B's position until time τ. The entries of $K_2^{(t)}$ are updated with each message sent. As an example, if B sends a message to B' at time $\tau' > \tau$, then B will update the entry for B' in $K_2^{(t)}$ to $\langle B', \tau' \rangle$.

Note that the information in this set can also be updated by indirect communication. For instance, let us suppose that B receives a message with content K_1'. If K_1' contains a more updated entry about B, then B knows that B' was informed about B's position indirectly by some agent. Beliefs in $K_2^{(t)}$ are used for pro-active messaging, as each agent is responsible for keeping other agents informed about itself. Therefore, the messaging overhead is reduced, as there is no need for other agents to send an `information request` message.

Agent B sends a message to B' with probability $\alpha \mathcal{H}(B_{K_2^{(t)}}(B'))$, where $\mathcal{H}(\cdot)$ denotes the normalized Shannon's entropy function and $\alpha \in \mathbb{R}^+$ is a scaling coefficient. This entropy is calculated over the probability distribution that describes the possible locations of B' according to B's beliefs $K_2^{(t)}$.

By making the assumption that every bus has the same probability of choosing all neighbouring stations, we can estimate the probability that a bus B' is heading to a station S_2 at time t by using its corresponding entry in the $K_1^{(t)}$ table of B – its partial knowledge about B'. This probability distribution was approximated by means of a random walk simulation on the map of the city, represented by the graph $\mathcal{M}$, and is part of the agent's beliefs. This process is illustrated in Figure 7. We denote this estimated probability by $\mathbb{P}(B'_{S_C} = S_2 | K_1^{(t)})$.



τ_0 $\tau_1 > \tau_0$ $\tau_2 > \tau_1$

Figure 3: Representation of the knowledge of the position of bus B' over time. Higher opacity indicates higher probability values. The probability distribution is assumed to converge to a uniform distribution in the limit after τ_0 while the entropy increases.

Note that communication is asynchronous and agents maintain their $K_1^{(t)}$ and $K_2^{(t)}$ beliefs until more up-to-date information is received. Using this method we resolve *inconsistencies* between an agent's beliefs and the real state of the environment. This clearly allows us to exhibit *focused addressing* and *node-available messaging* behavior [1, Ch. 8]. Note that our communication protocol allows for *graceful degradation* in face of outdated beliefs, as each agent is still able to operate in the environment and when necessary communicate with other agents. Each message sent between two agents bears a fixed cost and takes one tick to arrive from the sender to the recipient. The effect of the interaction between first and second order knowledge of two buses is represented in Figure 8 in section 4.

2.5 Collective Decisions

Passenger arrival is non-uniform and unknown by the agents. Moreover, the system is flexible to different passenger arrival distributions. To adapt our system to these properties, we allow agents to vote for adding new agents [1, Ch. 12]. We assign bus with id 24 as the mediator of the voting procedure. This bus is assumed to be present since the beginning of the simulation. Votes are submitted asynchronously and piggy-backed onto the regular messaging mechanism as described in section 2.4.

Each vote is accompanied with a timestamp, and the content of the vote represents the demand of the system for a new bus from the perspective of each agent. The unit of this quantity is the number of passengers that an agent expects cannot be transported under the current capacity distribution of the buses in the system. Therefore, the votes of each agent are part of its intentions and other agents' beliefs. We describe this process in detail below.

We define the matrix $\mathbf{P}^{(t)} = (\mathbf{P}^{(t)}_{S,S'})$ to be the number of passengers going from a S to S' at time t. Formally, $\mathbf{P}^{(t)}_{S,S'} \coloneqq \{P \in \mathcal{P} \colon S_P^{(t)} = S \wedge P_{S_D} = S'\}$.

Furthermore, let us denote the waiting time of each passenger at time t as $W_P^{(t)} \colon \mathcal{P} \times \mathcal{T} \to \mathcal{T}$ such that $W_P^{(t)} \coloneqq \max(0, t - P_{t_0})$. We can then express the cumulative waiting time of passengers between any two stations as $\mathbf{W}^{(t)} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$, where $\mathbf{W}^{(t)}_{s,s'} \coloneqq \sum_{P \in \mathbf{P}^{(t)}_{S,S'}} W_P^{(t)}$. The average waiting time in the entire system at time t is then computed as $\overline{\mathbf{W}}^{(t)} = \frac{1}{|\rho(t)|} \sum_{S,S' \in \mathcal{S}} \mathbf{W}^{(t)}_{S,S'}$.

The expected capacity $EC^{(t)}(S)$ and the leftover passengers $LO^{(t)}(S)$ at station S at time t dictate the content of each vote. Formally, we define them as follows:

$$EC^{(t)}(S) = \sum_{B' \in K_1^{(t)}} B'_C \cdot \mathbb{P}(B'_{S_C} = S | K_1^{(t)}) \tag{1}$$

$$LO^{(t)}(S) = \max\left(\sum_{S' \in \mathcal{S}} \mathbf{P}^{(t)}_{S,S'} - EC^{(t)}(S), 0\right) \tag{2}$$

$EC^{(t)}(S)$ computes the sum over the capacity of every bus in $K_1^{(t)}$ at station S at time t weighted by their probability of being at S. $LO^{(t)}(S)$ subtracts the number of passengers at S from the expected capacity at that station, thus representing the expected number of passengers who can not have be picked up at time t. Note that they are computed from the point of view of an agent, and are therefore subject to its beliefs.

The agent computes its preference by smoothing the total number of leftover passengers over all stations according to the average waiting time of the system $\overline{\mathbf{W}}^{(t)}$. The smoothing function converges to the total number of leftover passengers as the average waiting time increases, while it tends to zero for small waiting times. This preference is measured by number of passengers and it represents the vote of the agent.

Formally, $\mathcal{V} \coloneqq \{\langle i, V, \tau\rangle\}$ is a set of collected votes in which an element represents the vote V of the bus id i at time τ.

The mediator evaluates $\mathcal{V}$ at each tick. After a new bus B' is added to the system, a timeout period of 20 ticks is enforced during which no more buses are purchased. Note that B' communicates its presence to the mediator by sharing its beliefs. The timeout period inhibits the number of purchases, and gives each agent the opportunity to discover the existence of B' via communication.

The voting mechanism is defined as follows. If more than half of votes in $\mathcal{V}$ were submitted after the last purchase, the mediator considers purchasing a bus. It then computes the average preference, denoted as $\overline{V}$. If $\overline{V}$ is less than 6, then the purchase is cancelled. Otherwise, it purchases a bus whose capacity is closest to $\overline{V}$.

2.6 Multi-Agent Interaction

Given that we have already described the messaging protocol and the voting mechanism between the buses, we are left with the problems of picking-up passengers and assigning value to the possible actions that an agent can take at a particular situation. This determines where the bus will travel to in the next moment. For now we will concentrate on the latter problem and the pick-up policy will be described in Section 2.8.

Formally, we want to find a function $Q(Z_B^{(t)}, \mathcal{M}, \hat{S}_N)$ defined for the stations $\hat{S}_N$ which are adjacent to B_{S_C} according to the map $\mathcal{M}$. Q should satisfy that state-action pairs with a better impact in the performance of the system receive a higher score.

We define Q as the composition of two functions f and λ such that $Q(Z_B^{(t)}, \mathcal{M}, \hat{S}_N) = f(\lambda(Z_B^{(t)}, \mathcal{M}), \hat{S}_N)$. In this setting, λ is a feature extractor that receives raw state data and provides structured information regarding the situation of the bus in the system. Additionally, f is an evaluation mechanism that describes how adequate is to pick $\hat{S}_N$ given the current state of affairs. Further details for the functions λ and f are provided in Section 2.7.

Finally, we assume that there exists a monotonous transformation from the Q space into the real-world utility space, such that by optimizing Q we optimize the performance of the system in terms of the three criteria mentioned before.

This, of course, is a non-trivial problem due to the fact that the buses only have local and uncertain information available (in terms of their first and second order knowledge sets K_1 and K_2). Additionally, even in the case of perfect information, the decision making process in the system is asynchronous and decentralized. Therefore, we propose a game theoretic approach in which the buses play a Bayesian game every time they have to make a decision on where to go next [1, Ch. 11].

A Bayesian game [2] G is defined as a tuple $\langle \Pi, \Omega, \Theta, p, u\rangle$, where:

- Π is a set of players,
- $\Omega = \bigtimes_{i \in \Pi} \Omega_i$, where Ω_i is the set of actions available to player i,
- $\Theta = \bigtimes_{i \in \Pi} \Theta_i$, where Θ_i is the type space of player i,
- $p\colon \Theta \to [0, 1]$ is a common prior over types, and
- $u = (u_1, u_2, \ldots, u_{|\Pi|})$, where $u_i\colon \Omega \times \Theta \to \mathbb{R}$ is the utility function of player i.

Given the asynchronous nature of the decision process, we only consider the perspective of a specific player $B \in \mathcal{B}$ at time $t \in \mathcal{T}$. This implies that the perceived uncertainty of each possible outcome of the game is fully encoded in B's first order knowledge $K_1^{(t)}$.

Let $\theta \in \Theta$ be a type profile for the players in the game with $\theta = \left(\theta_1, \ldots, \theta_{|\Pi|}\right)$. For mathematical simplicity, we consider that the type space for a player i is given by the set of stations $\mathcal{S}$ and that the possible actions for each player is also $\Omega_i = \mathcal{S}$. Note that the buses can only make decisions to move towards adjacent stations. However, it is possible to construct an extended game in which the non-adjacent stations are included as possible actions such that there exists an isomorphism between the original and extended games' equilibria [2].

Consider $\mathbb{P}(B'_{S_C} = \theta_n | K_1^{(t)})$ as defined in Section 2.4. Now, we can define the function p as:

$$p(\theta) = \prod_{B' \neq B} \mathbb{P}(B'_{S_C} = \theta_n | K_1^{(t)}) \tag{3}$$

Agent i's *ex ante* expected utility in a Bayesian game under the (pure) strategy profile $\omega \in \Omega$ is defined as:

$$EU_i(\omega) = \sum_{\theta \in \Theta} p(\theta) u_i(\omega, \theta) \tag{4}$$

We refer the interested reader to [2] for a definition of solution concepts applicable to this sort of games. Let $sw(\omega, \theta)$ denote the social welfare for the action profile $\omega \in \Omega$ assuming player types $\theta \in \Theta$. Now, the expected social welfare is given by:

$$\begin{aligned} sw(\omega) &= \sum_{i \in \Pi} EU_i(\omega) = \sum_{i \in \Pi} \sum_{\theta \in \Theta} p(\theta) u_i(\omega, \theta) \\ &= \sum_{\theta \in \Theta} p(\theta) \left(\sum_{i \in \Pi} u_i(\omega, \theta) \right) = \sum_{\theta \in \Theta} p(\theta) sw(\omega, \theta) \\ &= \mathbb{E}_{\theta \in \Theta} \left[sw(\omega, \theta) \right] \end{aligned} \tag{5}$$

This means that we can aim to maximize the social welfare by a maximization of the expected welfare from each local decision of the agents given their current beliefs. This is possible for the system under the assumption that we control the agents and can ensure benevolent behavior among the buses in the transportation environment. Finally, this also leads to the formation and stability of the *grand coallition*, as the agents only care about the general system performance [1, Ch. 13].

2.7 Next Station Policy

As described in the Section 2.2, once a bus arrives at a station it drops off all of its passengers and updates its beliefs $K_1^{(t)}$ and $K_2^{(t)}$ based on its perception ρ and its inbox. Then, in order to determine its intentions, the agent evaluates a set of 4 different feature for every possible destination. Each score has been designed to represent a different view on the current state of the system that can lead to *reactive* and *pro-active* behaviours or ensure *cooperation* [1, Ch. 4]. The following paragraphs will define each feature created by the λ function.

Passenger Destination The first component of the compact state representation expresses how desirable is an adjacent station S_n as an intermediate stop for the passengers waiting at the current station S_c. This definition relies on a path optimality function $o(S_c, S_n, S_d)$ that defines how good is an adjacent station S_n as an intermediate stop for passengers travelling from S_c to their final destination S_d. The output of the o function is one if S_n is part of the shortest path from S_c to S_d, and linearly decreases to zero if travelling to S_n increases the path length from S_c to S_d by more than a fixed percentage (20% in the reported experiments).

This representation uses the passengers' waiting times as a weighting factor for path optimality. This weighting expresses the urgency of going to a particular destination. The total score for an adjacent station S_n evaluated at a station S_c at time t can be expressed as:

$$PD^{(t)}(S_n) = \frac{\sum_{S_d \in \mathcal{S}} o(S_c, S_n, S_d) \mathbf{W}^{(t)}_{S_c, S_d}}{\sum_{S_d \in \mathcal{S}} \mathbf{W}^{(t)}_{S_c, S_d}} \tag{6}$$

Note that this score representation promotes a *reactive* behaviour since it focuses on minimizing the waiting time of the passenger given the local view of the state.

S_d	$\mathbf{W}_{A,S_d}$	$o(A, S_d, B)$	$o(A, S_d, C)$	$o(A, S_d, D)$	$PD(S_d)$
B	10	1	0.5	0	$\frac{10\cdot1+8\cdot0.5+12\cdot0}{10+8+12} = \frac{14}{30}$ ✓
C	8	0.5	1	0	$\frac{10\cdot0.5+8\cdot1+12\cdot0}{10+8+12} = \frac{13}{30}$
D	12	0	0	1	$\frac{10\cdot0+8\cdot0+12\cdot1}{10+8+12} = \frac{12}{30}$

Table 1: Example of passenger destination score

Table 1 shows the computation of the passenger destination score PD for a bus at station A and the simple graph shown in Figure 4. Passengers in A have either B, C or D as their final destination. The total waiting time for the passenger directed to B is 10 while C and D have 8 and 12 ticks, respectively. Since stations B and C are connected and close to each other, choosing B as next station for passengers whose final destination is C is better than remaining at A. For this reason, the path optimality $o(A, C, B) = 0.5 = o(A, B, C)$. Since going to D would increase the travelling distance for the passengers that are directed to both B and C, this option is not taken into account and the path optimality is set to 0. Clearly, the path optimality for the next station is one if that station is the final destination of the passenger.

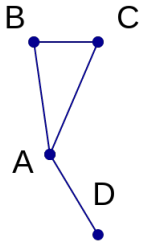


Figure 4: Example graph.

The total score for a station is then obtained by adding the waiting times multiplied by the respective values of optimality function. The obtained scores show that B attains the highest score since it is an optimal option for passenger directed there and, at the same time, makes passengers directed to C closer to their destination. Although D is associated with the highest cumulative waiting time, it is not selected since it would result in a sub-optimal choice.

Expected Load Factor The second component of the feature representation λ is based on the partial knowledge of the other agents' position $K_1^{(t)}$, the number of passengers waiting at the neighbouring stations S_n and the bus capacity C.

Based on the idea of *CDPS* [1, Ch. 8], we want to ensure an efficient collaborative behaviour between the agents, and thus they need to avoid responsibility overlaps and collisions. For this reason, we implement a *social norm* that discourages agents from travelling towards the same stations. The score for this component considers the expected number of leftovers $LO(S_n)^{(t)}$ as defined in Section 2.5. The expected load factor score EF for a bus B at station S_c and an adjacent station S_n at time t is defined as:

$$EF^{(t)}(S_n) = \min\left(\frac{LO(S_n)^{(t)}}{B_C},\ 1\right) \tag{7}$$

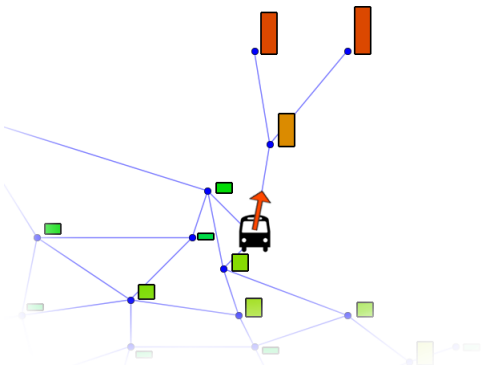


Figure 5: Effect of the total waiting time across the different stations in the map: buses are attracted towards regions with higher waiting times.

Waiting Time Decomposition In order to improve the *pro-activeness* of the agent, the third component of the state representation expresses the willingness of the buses to travel to a station where the total waiting time is high regardless its position on the map. For each adjacent station S_n, the waiting time decomposition score reflects how good is S_n as a next stop on a path towards the stations with the highest waiting time of the passengers. Formally it can be defined as:

$$WD^{(t)}(S_n) = \frac{\sum_{S_d, S_n \in \mathbb{S}} o(S_c, S_n, S_d)\mathbf{W}^{(t)}_{S_d,S_m}}{\sum_{S_n, S_d \in \mathbb{S}} \mathbf{W}^{(t)}_{S_n,S_d}} \tag{8}$$

Figure 5 shows the effect of this feature on the resulting behaviour of the agents by a simplified graphical representation.

Passenger Similarity Score The last component of the λ function has been designed as a compromise between a *reactive* and a *pro-active* behaviour. It defines a score based on the destination similarity of the passengers which are in the local view of the agent (the current and adjacent stations). The intended behaviour consists of forming clusters of passengers with the similar destinations in terms of their position on the map. This component ensures a global level of *coherence*.

The passenger similarity score PS for a station S_n given a bus B at station S_c and time t is given by:

$$PS^{(t)}(S_n) = \sum_{S_d, S'_d \in \mathcal{S}} \mathbf{W}^{(t)}_{S_c,S_d} \mathbf{W}^{(t)}_{S_n,S'_d} \kappa(S_d, S'_d) \tag{9}$$

with $\kappa(S_d, S'_d)$ as a kernel function representing the similarity between two stations.

2.7.1 Baseline Heuristic Model

Our baseline heuristic method is a linear combination of the features provided by the function λ. As the features are already normalized, in a standard setup we the f function is modeled as the sum of the features.

2.7.2 DQN

Reinforcement Learning (RL) has been extensively applied to MAS [3]. In an RL setting, the environment is modeled as a Markov Decision Process that may be fully or partially observable to each agent. Each action performed by the agent generates a feedback signal from the environment in terms of a reward function. The reward is a utility that indicates whether the action's outcome is in accordance with the agent's interests. Note that the reward signal may be delayed, and thus an agent may seek actions that provide long-term benefits.

We leveraged on the deep Q-learning (DQN) technique [4] to learn a Neural Network (NN) which approximated the Q function described in Section 2.6. This NN is the controller that each bus uses to select the next station. In this case, we chose the number of passengers delivered per step as the reward function. We trained the network with a replay memory of data collected from many simulations with multiple agents. Unstable training is a common issue during DQN execution. For this reason, we used target net learning to enhance the stability of the training procedure.

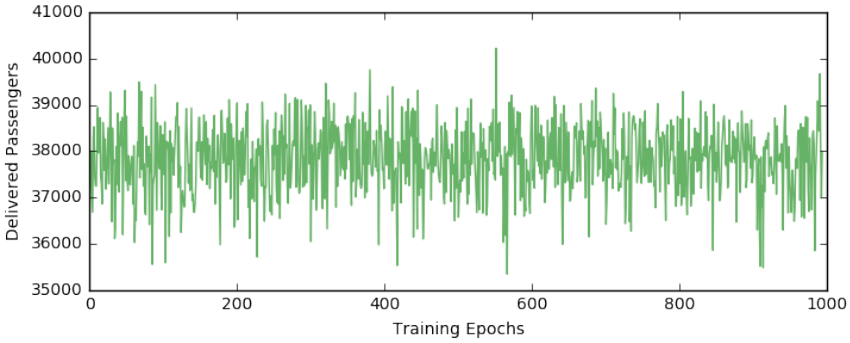


Figure 6: Number of passengers delivered across 1000 epochs

Training a DQN is a complex task which requires tuning a large number of (hyper)parameters, large training datasets and computational resources. While we experimented with several parameter settings, within the scope of this project we were not able to train a DQN that improves upon our baseline model. Figure 6 illustrates the non-improving behavior of the DQN across 1000 epochs in terms of number of passengers delivered.

2.7.3 Evolutionary Algorithms

As an alternative approach we decided to implement an evolutionary algorithm that learned an appropriate linear combination of the features provided by the λ function. An agent's genome encodes the bus type (small, medium, large), the weight associated with each component of the feature vectors, and the parameter α that regularizes the communication frequency.

This allowed us to create a *subsumption* architecture, where the priority of a behavior is dictated by the relative weight of its corresponding feature. For instance, a bus with a large weight for the waiting time decomposition

would travel directly towards stations with high waiting times rather than avoiding collisions with other buses. This shows how the waiting time decomposition could inhibit other behaviors of the bus [1, Ch. 4].

In this approach we defined a crossover operation that randomly exchanges or averages the corresponding genome components from the parents. The mutation operation is defined by adding a Gaussian noise vector with different variances per component to the genome of the bus. The parent selection was made by random sampling without replacement according to the fitness of the individuals.

2.8 Pick-up Policy

Once the bus arrives at its destination it drops off all the passengers and starts a new selection in order to optimize the allocation of the seats, see Figure 2. Since the destination has already been fixed according to the approaches defined in the previous section, the bus has to decide which passenger to pick up for its trip. As the resources are limited, the passenger have compete for the available seats. This competition is organized as a *sealed-bid first-price auction* where each passenger can place a bid according to his value function [1, Ch. 14]. Note that in this setup we assume that each passenger is bidding exactly his true valuation function that is defined similarly to the passenger destination score mentioned in Section 2.7. Formally the valuation function v for a passenger P waiting at station S_c for a bus travelling to S_n as time t is defined as:

$$v^{(t)}(P) = o(S_c, P_{S_D}, S_n) W_P^{(t)} \tag{10}$$

As the buses want to maximize the social welfare, they will select the highest bids to fill their capacity. Note that since the path optimality function can assume a value of zero, passengers will assign zero as the value for destinations that excessively deviate from their optimal path. Even if the buses have some capacity left, they do not pick up passengers that are bidding zero values.

3 Implementation

We use Python's `Keras` and `tensorflow` packages to implement the DQN. The parameters of the final network's architecture are two fully connected hidden layers of length 200 and 100 respectively, with rectified linear unit activations. The output layer is a scalar with a linear activation, which represents the score given a state representation vector of length 144. We use the ADAM optimizer [5] for batch stochastic gradient descent and the mean-squared error function. For specific parameter values, we refer the reader to the source code of this project.

4 Results

We present our results in this section. Within the given timeframe of the project and our computational resources, we were not able to adequately train a DQN model. As discussed in Section 2.7.2, we omit it from this section. Overall, the training procedure of the neural network did not converge under different parameter settings and was never able to outperform the baseline heuristic model.

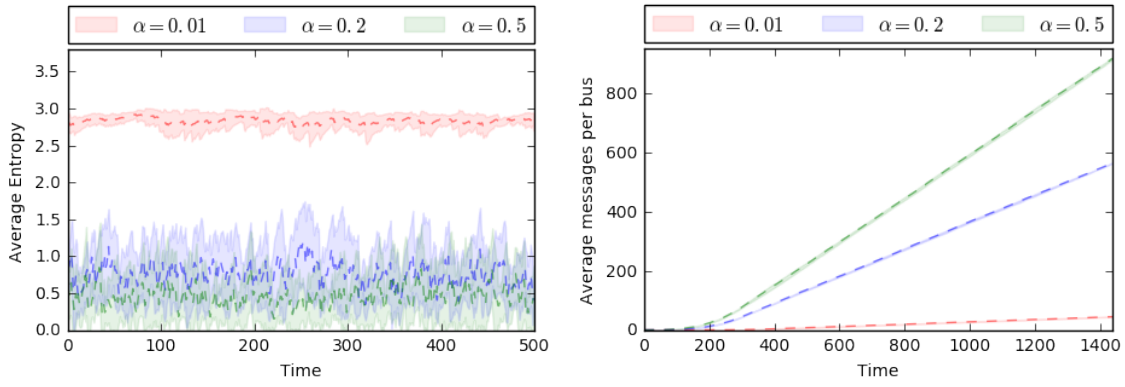


Figure 7: Graph showing the effect of α on the average entropy of $K_1^{(t)}$ and number of messages sent per bus.

Figure 7 illustrates the effect of the parameter α in the messaging protocol. Higher values of α result in more communication between the agents, which allows them to maintain a low level of uncertainty in their beliefs throughout the simulation.

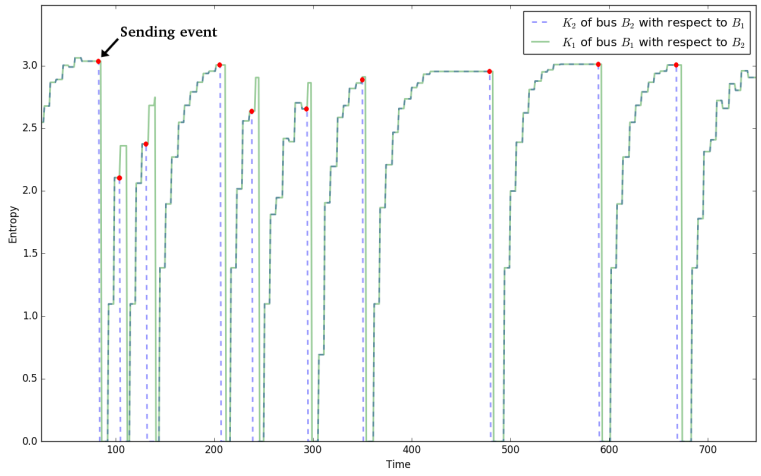


Figure 8: Decreasing effect of sending a message on the uncertainty (entropy) in the beliefs of the buses.

Figure 8 visualizes the uncertainty estimation of two agents' beliefs in a simple scenario where they send messages to each other. We can observe that each message decreases the entropy across their beliefs. The decrease in entropy is seen in the $K_2^{(t)}$ beliefs of sender (with respect to the receiver), and in the $K_1^{(t)}$ beliefs of receiver (with respect to the sender).

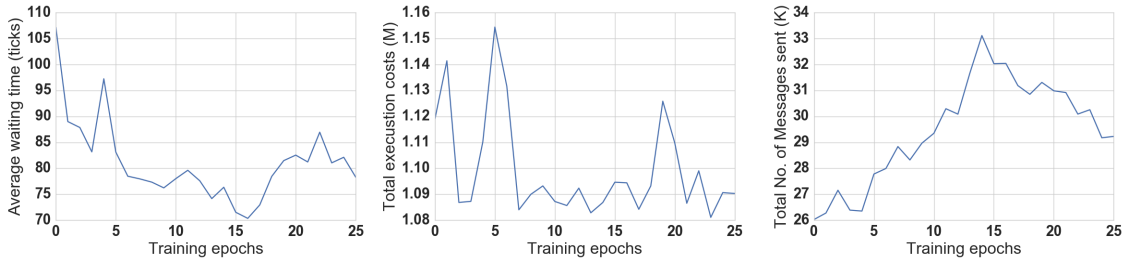


Figure 9: Performance of the evolutionary model across training epochs.

Figure 9 displays the performance of the evolutionary algorithm. The best overall model is obtained at training epoch 16. This model is compared with the baseline heuristic model in Table 2. The evolutionary model is initialized using the same baseline heuristic. We can observe that it significantly improves the average waiting time and execution costs at the expense of sending more messages. This is the expected behavior, as there is a trade-off between the the objectives of the system. Sending a larger number of messages improves the *coordination*, which results in a better solution quality, i.e. the *coherence* improves.

Model	Average Waiting Time	Total Execution Costs (M)	Total Messages Sent (K)
Baseline	105.86	1.249	**27.45**
Evolution-16	**71.37**	**1.0943**	32.16

Table 2: Comparison of performance for baseline heuristic and evolutionary models

5 Conclusions

As discussed in Sections 2.7.2 and 2.7.3, we believe that our two approaches using DQN and evolution are sound, and prior to conducting our experiments we expected significant and stable improvements over the baseline model. The results, however, do not support our hypothesis as the DQN model was not able to outperform our baseline heuristic model. We did observe, however, that by using evolution to tune the mixing coefficients of the subsumption architecture of our baseline model, we are able to obtain better performance with respect to the average waiting time and total execution cost.

In this research, we used a *hybrid* approach to modelling our agents. Our baseline method uses knowledge-based heuristics and decision functions to determine the various behaviours of our agents discussed in Section 2.2 and illustrated in Figure 2. Our DQN and evolution models use data-driven techniques to learn different representations of the most complex behaviour of the agent, namely picking the next destination. The motivation behind using data-driven approaches were to build more complex models of agent behavior and improve the generalization capability of the baseline model. A common difficulty with data-driven approaches is that they are difficult to understand and debug, often requiring lots of data and training iterations to tune (hyper) parameters. This was not possible within the scope of the project and the computational resources available.

On the other hand, the proposed methodology for agent interaction and communication as well as the voting mechanism have proven to be sound and useful towards the solution of the problem.

Further explorations of this task could be approached by considering different state representations (for instance, ensuring that the Markov property is satisfied), different belief systems, long-term planning and the introduction of domain-specific techniques related to transportation systems.

References

[1] M. J. Wooldridge, *An introduction to multiagent systems*. John Wiley, 2 ed., 2009.

[2] K. Leyton-Brown and Y. Shoham, "Essentials of game theory: A concise multidisciplinary introduction," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 2, no. 1, pp. 1–88, 2008.

[3] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *Trans. Sys. Man Cyber Part C*, vol. 38, pp. 156–172, Mar. 2008.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.