UTRECHT UNIVERSITY

BACHELOR THESIS
BSC ARTIFICIAL INTELLIGENCE

Modeling normative behavior and enforcement in a multi-agent system based on Dutch Public Transport

---

Author:
Rick MANK

Student Number:
5715660

Supervisor:
Dr. Natasha ALECHINA

Second reader:
Dr. Rick NOUWEN

A 7.5 ECTS Thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in

Artificial Intelligence

at the

Faculty of Humanities

June 25, 2020

Abstract

Rick MANK

Modeling normative behavior and enforcement in a multi-agent system based on Dutch Public Transport

We live in a normative culture in which people make decisions every day to either follow or break rules and norms. An interesting example of choosing behavior can be found in the public transport system, which as it is a semi-closed system lends itself to being modeled. This human behavior can be modeled to predict and influence the way in which agents and people alike make desirable decisions.

This thesis will answer the question 'how can enforcement strategies for public transport be optimized by making use of learning agents'. To answer this question I first had to research different decision-making strategies such as decision theory and reinforcement learning. With this I gained a better understanding of the normative culture of choosing agents, with which this thesis aims to predict norm abiding and norm breaking behavior in a stochastically modeled environment.

Contents

Chapter 1

Introduction

1.1 Introduction to norms

We live in a society where many of our actions are regulated by rules and norms. Imposing norms works in such a way that individual agents are provided certain obligations and prohibitions, which can be monitored by a normative organization.[1,2] If an agent fails an obligation or commits a prohibited action, the organization will impose a sanction on the agent. Human behavior is interesting when norms are involved as humans are *norm-autonomous*, which means they are able to make decisions about compliance in regards to norms.[14] Different individuals behave differently depending on what norm is involved and depending on their values and needs.

Suppose an individual agent aims to obtain a maximal reward. This could occasionally result in a conflict in the agent: risking a sanction by violating a norm or losing out on a potential higher reward by complying. In certain real-life scenarios, such as public transport, managing to defy a norm undetected yields a higher reward than complying to said norm. In real life many different factors contribute to decision making, such as morality, predicted rewards, risk assessment and risk aversion. Because of the abstract nature and variance in morality across a population this is hard to model, therefore this paper will strictly focus on rewards and punishments. As it is the agent's goal to obtain a maximal reward, in a normative society where the highest reward can be obtained by defying norms I suspect the agents to attempt to defy the norm without being sactioned. On a similar note, the priority of the normative organization will be to catch and punish agents that defy the norm.

Norms lay out conditions that obligate or prohibit certain behavior. For example, in traffic it is prohibited to run a red light. In other words, a person is obligated to stop for a red light. Agents that violate this norm must pay a fine if they have been caught running a red light, while agents that comply keep their money in their pockets but must wait until the light goes green. The agent has to decide whether the risk of getting caught (not taking into account the danger of running a red light) is worth the reward of not having to wait. The agent must assess how it values the various relevant factors in the situation, like time, money and safety. An agent that is able to reason this way with respect to norms is called a *norm-aware agent*.[1]

If we assume the agent has perfect information about the sanction size and monitoring frequency, it will simply have to use basic math to calculate whether the reward is higher when complying to the norm or not. Similarly, if the normative organization has perfect information about the moments of norm violation, they would succeed in catching every violating agent. In the real world both scenarios are not the case. It is unrealistic that every violation of a norm goes detected by a normative

organization. In other words: it is reasonable to assume the probability of detecting a norm violation, the *enforcement intensity*, is less than one.[10,11] Monitoring is often done by random sample tests, because it would be impossible to supervise every instant due to limited resources.[11] The agents having perfect information is unrealistic as well. The enforcement intensity is often unknown, precisely to make sure the monitoring can't be evaded.[11]

In a normative situation, we can view both parties as learning agents. The normative organization could theoretically learn what the best moments would be for employing enforcement, based on trends in the moments or frequency of norm violation by agents. Similarly, the agents could form a policy to determine whether to comply to the norm or not. In this thesis, I will investigate two methods for an agent to determine their behavior: based on decision theory and based on Reinforcement Learning (RL). The first method will be covered purely theoretically, while I will apply Reinforcement Learning to a programmed simulation.

1.2 Introduction to the problem

The problem I will apply the aforementioned techniques to is public transport. In 2006, around 3.5% of public transport users in The Netherlands traveled without having paid for a valid ticket.[4] This not only poses a problem for financial reasons, but a problem for safety reasons as well. A research conducted in 2014 on violence towards Dutch train personnel showed that almost 70 percent of the cases were caused by someone who used transport without paying.[6]

Making use of public transport is a normative situation, with passengers as agents and ticket inspectors as the enforcing party. To travel by public transport, a traveler must be in possession of a valid ticket, or must be checked in with their public transport card. This is the only obligation present in this normative situation. Travelers pay a small sum of money to be allowed to travel, and if they defy the norm they risk a fine of a much higher sum. The amount of both prices is readily available information for travelers, so it is safe to assume the traveling agents have perfect knowledge of the rewards.

Not every environment of a normative situation can be perfectly monitored.[2,11] The cost for monitoring the ideal norms could be too high, or simply impossible. As mentioned in 1.1, the enforcement intensity in public transport lies below 1. No train or bus has a ticket inspector present at all times, there are simply not enough resources available to achieve that. The normative organization employs ticket inspectors at random, to catch and fine people that have not paid for their travel. In the remainder of this thesis I will refer to this problem as the *public transport problem*.

1.3 Aim and relevance

The aim of this project is to continue on the work done by Li *et al.*[10] This thesis will be built on the paper of Li *et al.* and adopt a similar approach. I hope to gain understanding of different strategies that agents and enforcing parties can develop in a normative situation. Ultimately the goal is to let the traveling agent learn from the normative organization via reinforcement learning algorithms. This could give insight in a potential optimization of enforcement strategies for fields such as public transport. In other words, I hope to answer the following research question: "How

can enforcement strategies for public transport be optimized by making use of learning agents?"

This thesis is relevant in the field of Artificial Intelligence in multiple ways. Modeling human behavior and creating rational agents are both fundamental approaches to AI according to Russell and Norvig.[12] Moreover, even though programs cannot really make decisions the way humans do, but rather act according to the decision-making rules embedded by the programmer, decision theory is an integral subject within AI. Human reasoning needs to be analysed to be able to model decision processes for computer programs.[13].

Finally, Reinforcement Learning plays a big role in agent decision-making. In the last twenty years it has risen as a method for creating self-learning programs, which are useful for simulating all sorts of real life scenarios and processes. By using rewards and punishments without specifying exactly how a task should be done, the agent learns a policy to base decisions on.[7]

In this thesis I combine these three elements and apply them to a problem with societal relevance. The results of this research could provide insight in enforcement strategies that can be used by public transport organizations.

1.4 Structure of this paper

In the next chapter (Chapter 2) of this thesis I will give a formal definition of norms, after which I will shed light on the decision-making techniques I mentioned above. Decision theory will be covered theoretically and applied to the public transport problem in section 2.2. The results of this analysis are used to check whether an optimal policy can be learned through reinforcement learning. In section 2.3 I will give background information about RL and the RL-method used: Q-learning. In Chapter 3 the process of implementing the RL algorithms is described, as well as the results that occur. Those results will be discussed in Chapter 4, where I will reflect on the results and the research question posed in the introduction.

Chapter 2

Approaches to the problem

2.1 Theoretical background on norms

This section provides some background information on norms. To be able to program the behavior of a traveling agent, their behavior must first be denoted in a sufficiently abstract way. For this purpose I decided to adhere to the formal definition for norms used by Li *et al.*, as it is an intuitive representation of the elements needed to describe a norm. (see Definition 2.1.1).[10]

Definition 2.1.1 (Norms).
A norm is a tuple $\langle \delta, G, \phi, \psi, \rho \rangle$ where:

- $\delta \in \{\text{obligation, prohibition}\}$ is the deontic modality;
- G is a set of agent roles to which the norm applies;
- ϕ is the activation condition, which induces a set of states S_ϕ such that $S_\phi = \{s \mid s \in S \wedge s \vDash \phi\}$;
- ψ is the normative condition, which induces a set of states S_ψ such that $S_\psi = \{s \mid s \in S_\phi \wedge s \vDash \psi\}$;
- ρ: $S \rightarrow \mathbb{R}$ is a function that specifies the penalty for violating the norm in a given state ($\rho(s)$ returns the penalty to be paid in s).

The definition works as follows: a norm $n = \langle \delta, G, \phi, \psi, \rho \rangle$ activates in a state where the activation condition holds ($s \in S_\phi$) for an agent a if the role of the agent ($role(a)$) is a role to which the norm applies ($role(a) \in G$). In this case the norm to check in before traveling with public transport applies only to the role *Traveler*: $G = \{Traveler\}$. The norm is obeyed if the normative condition ψ holds in s (in the case of obligations) or does not hold in s (in the case of prohibitions). Otherwise the norm is violated in s, and the agent must pay a penalty $\rho(s)$ in s. In the public transport problem there exists an obligation for the traveler, namely to check in. The normative condition ψ holds when the traveler obeys the norm, so the normative condition ψ = "the traveler has checked in".

Whether the activation condition ϕ holds in a state or not is determined by the normative organization. In states where it does hold, the normative organization is responsible for determining whether the norm is obeyed or violated. In situations where the norm is violated, the normative organization imposes the appropriate penalty. The set of violated norms can be defined as follows, in Definition 2.1.2, where N_s^- is a set of norms:

Definition 2.1.2 (Violated Norms).
$N_s^- = \{\langle \delta, G, \phi, \psi, \rho \rangle \in N \mid \delta = \text{prohibition} \wedge s \vDash \psi\} \cup$
$\{\langle \delta, G, \phi, \psi, \rho \rangle \in N \mid \delta = \text{obligation} \wedge s \nvDash \psi\}$

This entails that a norm $n = \langle \delta, G, \phi, \psi, \rho \rangle \in N$ can be violated in two possible ways: if the norm is of a prohibitive nature (δ = prohibition) and the action prohibited by the norm is performed nonetheless ($s \vDash \psi$), and if the norm is of obligative nature (δ = obligation) and the action obliged by the norm is not performed ($s \nvDash \psi$).[10]

2.2 Decision Theory

In this section I will regard the traveling agent in the public transport problem as a decision-theoretic agent. By applying the normative approach described in section 1.1, I can reason about a fundamental principle of decision theory, which is maximizing expected utility.[12]

If the agent is to determine the maximal expected utility for its actions, a *utility function* $U(s)$ is needed, which assigns numerical values to states based on their desirability. The traveling agent does not yet know what state (monitored or non-monitored) it will end up in after making the decision to comply to the norm or not. This means probabilities need to be taken in account. The probability of the outcome s', given evidence observations e and given that action a is executed, is described in Definition 2.2.1, where $Result(a)$ is a variable that stands for the outcome of executing action a.[12]

Definition 2.2.1 (Outcome Probability).
$P(Result(a) = s' \mid a, e)$

One action could lead to multiple different outcomes, so we need to calculate the average utility value of each outcome and multiply it with the probability of that outcome occurring. This gives us the expected utility of an action given the evidence, $EU(a|e)$ as shown in Definition 2.2.2:

Definition 2.2.2 (Expected Utility).
$EU(a|e) = \sum_{s'} P(Result(a) = s' \mid a, e)U(s')$

I want the agent to act rational, and choose the action with the *maximum expected utility* (MEU) from its set of possible actions, as defined in Definition 2.2.3:

Definition 2.2.3 (Set of possible Actions).
$a^* = \arg\max EU(a \mid e)$

The traveling agent in the public transport problem receives negative rewards in the form of ticket fees and fines. I assume traveling by public transport yields a large positive utility for a traveler, as the traveler wants to end up in a certain destination.

Because of this I set the maximal utility to 100, from which potential fees and fines are subtracted. The utility of 100 is only received when the traveler does not check in and the normative organization does not enforce the norm. For the costs of ticket fees and fines I used proportions similar to real-life costs. The utility values I use for calculating the MEU are shown in Table 2.1.

	Checked In	**Not Checked In**
Monitoring	95	50
Not Monitoring	95	100

TABLE 2.1: This table shows the rewards an agent receives in different situations in the public transport problem.

The decision problem can now be represented visually in a *decision tree*[13] (see Figure 2.1). The root node is a box, which signifies a choice is to be made at that moment in the decision process. At this point our traveler makes the decision whether to check in or not. The first child node then is a circle, which means a probabilistic event takes place there. The normative organization monitors whether the agent entails the normative condition ψ (see Definition 2.1.1) with probability p. In other words, p is the enforcement intensity. Lastly, each terminal node shows the utility scores for the traveling agent in each scenario.

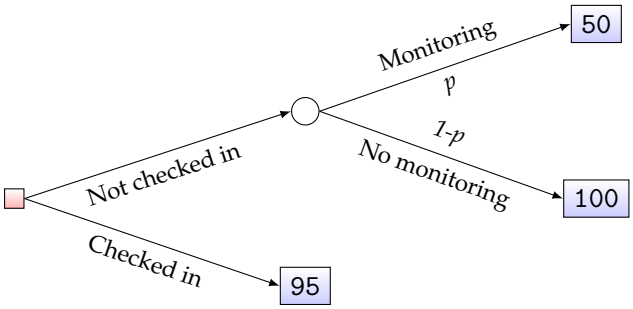

FIGURE 2.1: A decision tree for the traveling agent.

These utility scores allow the expected utility to be calculated for each action given the evidence: $EU(\text{Checking in} \mid e) = 95$ and $EU(\text{Not checking in} \mid e) = 50p + 100(1-p) = -50(p\text{–}2)$. It is expected that the agent complies to the norm and checks in when the enforcement intensity is high. When the enforcement intensity is low however, not checking in should have a higher expected utility. This means there must be a value for the enforcement intensity where the agent changes his preference. By plotting the expected utility functions against the enforcement intensity, it is shown that with these parameters this critical value is 0.1 (see Figure 2.2).

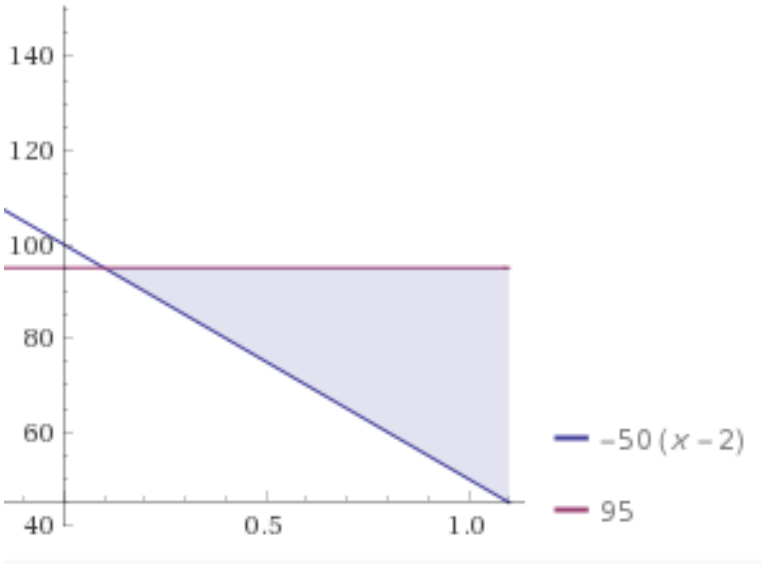

FIGURE 2.2: plot showing both expected utility functions plotted against the enforcement intensity.

2.3 Reinforcement Learning

As shown in Section 2.1, a traveling agent will gain a higher expected utility from checking in as opposed to not checking in, when the probability of monitoring is 0.1 or higher with the reward parameters from table 2.1. However, it is unrealistic for someone that uses public transport in real life to know in advance the probability for enforcement. As mentioned in Section 1.1, the enforcement intensity is often kept unknown so that agents cannot easily evade the enforcement.

It is not unrealistic for someone who travels by public transport often to become aware of possible patterns in enforcement, after a certain amount of time. This is where reinforcement learning becomes applicable to the problem. As stated by Kaelbling *et al.*: *"reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment."*[7] The agent receives information about the environment through perception and is able to influence the environment through actions. The interaction between agent and environment happens in steps, in each step the agent receives a certain input about its current state of the environment. Then the agent chooses an action, which in turn yields a certain numerical reward for the agent. The agent should aim to choose actions that tend to maximize the total reward, which it can learn over time via trial and error.

To formalize the structure of RL, I formulate it as a *Markov Decision Process*[10] (MDP) (see 2.3.1). This is possible because the problem is a sequential decision problem with a Markovian transition model. The probability of transitioning from state s to s' does not depend on earlier states the agent has been in, it solely depends on state s itself.[12]

Definition 2.3.1 (Markov Decision Process).
An MDP is a tuple $\langle S, A, P(s \mid s, a), R \rangle$, where:

- S is a set of possible states. For all states in discrete time steps, $st \in S$.

- A is a set of possible actions, where for all actions a possible in a given state, $a \in A$.
- $P(s' \mid s, a)$ is the probability of moving from state s to s' when executing action a, such that $\sum s' P(s' \mid s, a) = 1$ and $P(s' \mid s, a) \geq 0$.
- R is the reward function, mapping from states to reward values. $R : S \to \mathbb{R}$.
- γ is the discount factor, $0 \leq \gamma \leq 1$. The discount factor determines the importance of future rewards.

To aim for the maximal reward, the agent should develop a policy π, that maps from states to actions. Via this policy the agent knows what action to take in every state of the environment. To learn this policy, assigned to every state is a numeric value that indicates the expected reward received were the agent to follow that state. An optimal value function V has to be learned to assign these values to states, which is defined in Definition 2.3.2.

Definition 2.3.2 (Optimal Value Function).
$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P(s' \mid s, a) V^*(s')$

From this function V, the function for finding the optimal policy can be derived, as shown in Definition 2.3.3. The optimal policy π^* is defined as the best action a such that the expected total reward is maximized in state s.

Definition 2.3.3 (Optimal Policy).
$\pi^*(s) = \operatorname{argmax}_{a \in A} \sum_{s' \in S} P(s' \mid s, a) V^*(s')$

I will apply the RL method Q-learning on the public transport problem, where the formulas described in this section will be used. In the next subsection I will give a background on the method, and then present the implementation in Chapter 3. The goal is to allow the traveling agent to learn a policy for checking in based on fixed and stochastic enforcement patterns.

2.3.1 Q-learning

Q-learning is a commonly used method for reinforcement learning.[9,10] It is a method that learns utilities bound to actions, rather than just learning utilities. To describe the value for performing action a in state s, I use the notation $Q(s, a)$. Definition 2.3.4 shows the relation between Q-values and utility.

Definition 2.3.4 (Utility).
$U(s) = \max_a Q(s, a)$

The idea for Q-learning is to assign a Q-value to each action that can be chosen in a state, for every state. After choosing an action a in state s, the agent ends up in state s' and receives a reward R. Based on the reward, the Q-value $Q(s, a)$ gets updated

via the function described in Definition 2.3.5 with learning factor α and discount factor γ to account for temporal difference.

Definition 2.3.5 (Updating the Q-value).
$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_a Q(s',a') - Q(s,a))$

A well-known problem in RL is finding a balance between exploration and exploitation.[10,12] If the agent always chooses an action according to its learned policy, in other words exploiting the policy, the agent might stay at a local optimum in regards to the policy. Too much focus on exploitation might prevent the agent from finding possible better actions. To counter this, exploration is essential. This means the agent randomly picks an action without taking the policy in account. It is obvious to see that too much focus on exploration would not work either, as in that case the agent would not use any information it has learned. Like Li *et al.* I use an epsilon-greedy approach to balance exploration versus exploitation.[9,10] This means the agent chooses an action based on its learned policy most of the time, but it chooses a random action (exploration) with probability ϵ, which is set to 0.1.

Algorithm 1 shows pseudocode for the algorithm used in the RL experiments. Each episode the agent loops over the seven states representing a week, where each day has a corresponding state s.The epsilon-greedy strategy then decides whether the agent chooses a random action, or an action based on the policy. The reward based on the chosen action is then given by the reward function. (see Algorithm 2) The optimal next state is retrieved, which is always the next day as that is the only state the agent can go to. The only exception is when the agent is on day seven, then the episode is finished. The last step is updating the value function, which is done using the function shown in Definition 2.3.5.

Algorithm 1 Pseudocode Q-learning

```
1:  procedure TRAIN
2:      for each episode ∈ maxEpisodes do
3:          for each day ∈ week do
4:              s ← GetState()
5:              if isExplorations(s) then
6:                  a ← random a ∈ A
7:              else
8:                  a ← π(s)
9:              r ← getReward(s, a)
10:             s* ← getOptimalState(s)
11:             V(s) ← V(s) + α · [r + γ · V(s*) − V(s)]
12:             s ← nextState()
```

Algorithm 2 getReward(s, a)

```
1: procedure GETREWARD
2:     if a = CheckIn then
3:         r ← costReductionForCheckingIn
4:     else
5:         if normEnforced(s) then
6:             r ← penaltyForNotCheckingIn
7:         else
8:             r ← R(s)
9: return r
```

Chapter 3

Algorithm Implementation and Results

In this chapter I will use the theory and formulas described in Chapter 2 to program a traveling agent in a public transport scenario, using the programming language Python. I modified a Q-learning algorithm found online for another RL problem, the Cliff-Walking problem[5], to make it applicable to my public transport problem (see Appendix A for the code). In the first experiment, the agent will learn a policy based on fixed enforcement patterns. This will be covered in section 3.1. In section 3.2 I will describe the experiment with stochastic enforcement, from which the agent is to learn a policy as well.

3.1 One-dimensional array-based RL with fixed enforcement

The public transport problem is similar to the Parking World scenario researched by Li *et al.*[10] To illustrate variable norm enforcement, Li *et al.* used reinforcement learning on an agent in a 5x5 grid of cells. The cells represented spots where the agent could travel to with their car. One of the cells is a legal parking cell, and one is an illegal parking cell. The reward for parking on the legal parking cell is positive, whereas the reward for parking on the illegal cell depends on the normative organization. When the norm is enforced, the agent receives a large negative reward, but when the norm is not enforced it receives a large positive reward. This means the probability distribution for a positive and negative reward in the illegal parking spot depends on the enforcement intensity. This enforcement intensity is not known to the agent initially, but the idea is for the agent to learn it. It is expected that a rational choosing agent refrains from choosing an action that is likely to lead to a penalty. On the other hand, if the probability of getting caught parking illegally is low, the agent will likely aim for the highest reward by parking illegally more often. As discussed in Section 2.1, this means there exists a critical value for the enforcement intensity where the probability of getting caught parking illegally is too high for parking illegally to be a viable strategy to get a maximal reward. Figure 3.1 shows the average utility for both parking strategies plotted against the enforcement intensity from the experiments of Li *et al.*[10]

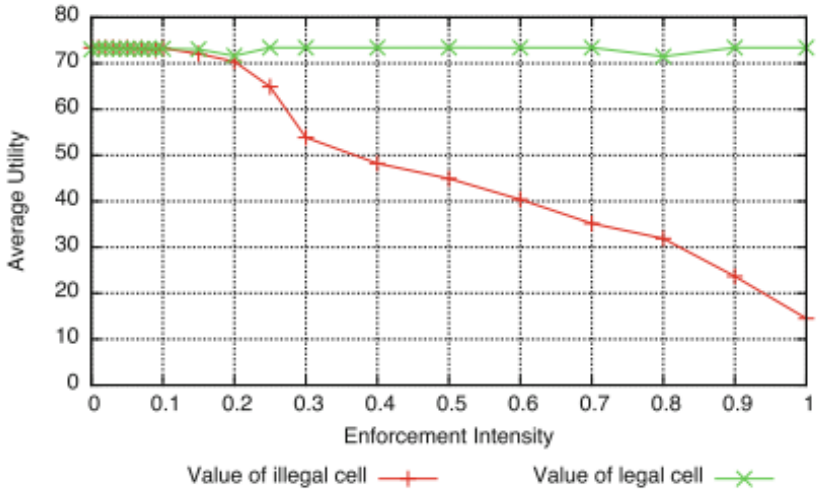

Fig. 3. Learned utilities for differing enforcement intensity (Q-learning)

FIGURE 3.1: Learned utilities for differing enforcement intensity (Li *et al.* (2015)).[10]

As can be seen, the critical value for enforcement intensity seems to be around 0.15, when using the parameters Li *et al.* used. The graph shows the expected behavior. Where Li *et al.* used a 5x5 grid of cells to simulate possible parking cells, the public transport model can be much simpler. It does not need to be grounded in two dimensions, it can be a one-dimensional array instead. To represent a week, I use an array of length 7 and the agent starts on the first day: (0,0). Every step the agent makes a decision to check in that day or to travel without paying, and then receives a reward based on their action and based on the normative organization. The Q-value for the relevant state and action is updated, and the agent then moves one 'cell' to the right. I use the same rewards here as in section 1.2 (see Table 2.1).

Prior to a run of the experiment, an *enforcement map* is generated which is an array of length 7 as well, consisting of 0s and 1s. This map determines at what days monitoring is present (represented by a 1 in the map), or not (represented by a 0 in the map). This allows the model to retrieve the correct rewards. The enforcement intensity can be increased or decreased per run, causing the number of days with monitoring to increase or decrease respectively. For example, an enforcement intensity of 0.5 means that each spot in the enforcement map has 50% chance of being a 0, and 50% chance of being a 1. In other words, each day has 50% of having enforcement being employed. In this part of the experiment I keep the enforcement fixed for every episode to see whether the agent is able to learn a policy. An example of an enforcement map with an enforcement intensity of 0.5 could be [0, 1, 1, 0, 1, 0, 0], which stays the same for every week (episode). This causes the traveling agent to receive a fine on Tuesdays, Wednesdays and Fridays, were the agent to defy the norm on those days.

It is expected that the agent is able to learn the pattern, and I am aware this is an unrealistic strategy of enforcement. However, it could give insight in behavior with respect to enforcement intensity nonetheless. Also, this model is easily expanded on with more complex, more realistic enforcement strategies as will be discussed further in Section 3.2.

I varied the enforcement intensity from 0.0 to 1.0 with a step size of 0.1. The agent was run ten times with 100.000 episodes per run, and after each run the average utility (Q-values) for checking in and not checking in were calculated. Then the average utilities were calculated over the ten runs for each value of enforcement intensity. The results of Q-learning on the public transport problem with fixed enforcement are shown in Figure 3.2.

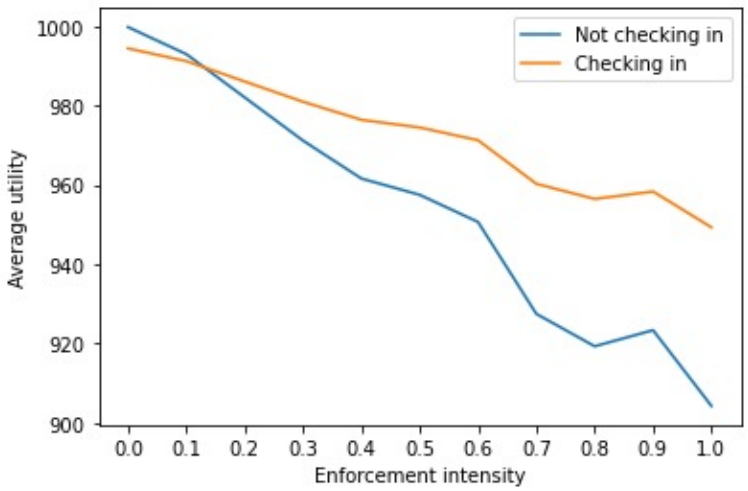

FIGURE 3.2: Results from Q-learning with fixed enforcement

As can be seen in Figure 3.2, the critical value for enforcement intensity is 0.13. After that point the average utility for not checking in drops below the average utility for checking in, and decreases at a faster rate.

3.2 One-dimensional array-based RL with stochastic enforcement

In Section 3.1 the agent learned a policy that caused it to prefer not checking in when the enforcement intensity is below 0.13. As the enforcement intensity increases, the agent becomes less likely to defy the norm. The enforcement map that determines which days of the week are monitored was fixed during all episodes of one run. As I stated in Section 3.1, most real life enforcement strategies will not be fixed, as those would be too easy to bypass. In this section I will apply the Q-learning algorithm on the public transport model with a stochastic enforcement strategy.

In this experiment, the enforcement map will not be constant for each episode. The enforcement intensity still influences the number of days where enforcement will take place, but the pattern will change every week. For example: an enforcement intensity of 0.1 can cause the enforcement map to be [1, 0, 0, 0, 0, 0, 0]. The next week still has an enforcement intensity of 0.1, but the enforcement map is generated anew. There is a less obvious pattern to enforcement in this experiment, but the traveling agent should still learn a policy based on its rewards.

The set up of this experiment is the same as in 3.1. Once again I varied the enforcement intensity from 0.0 to 1.0 with a step size of 0.1. The agent was run ten times with 100.000 episodes per run, and after each run the average utility (Q-values) for

checking in and not checking in were calculated. Then the average utilities were calculated over the ten runs for each value of enforcement intensity. The results of Q-learning on the public transport problem with stochastic enforcement are shown in Figure 3.3. The critical enforcement intensity value is shown to be 0.05 for stochastic enforcement.

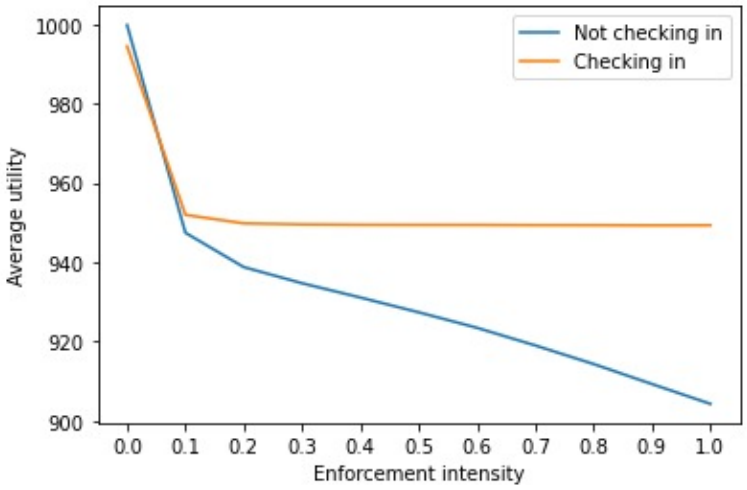

FIGURE 3.3: Results from Q-learning with stochastic enforcement

Chapter 4

Discussion

In this Chapter I will discuss the results of my experiments, as well as shortcomings of the project. Additionally, I will propose options for potential further research, and present the conclusion of this thesis.

Q-learning with fixed enforcement

When comparing the results from Q-learning with fixed enforcement (see Figure 3.2) with the results from Li *et al.* (see Figure 3.1), it can be seen that both graphs portray similar behavior. A clear point for the critical value of enforcement intensity is visible, which shows the agent managed to learn a policy which causes it to defy the public transport norm at low enforcement intensity. That preference quickly decreases after the value for enforcement intensity reaches the critical point of 0.13. The agent has learned it yields a higher reward if it complies to the norm from there on. The average utility for checking in decreases as well, which means increases in the enforcement intensity still have an effect on the average utility of checking in. The result is significant nonetheless, as the average utility for checking in is higher than the average utility for not checking in when the enforcement intensity is 0.13 or higher. The prefered policy is the action with the highest average utility relative to the other actions in the action space, regardless of increases or decreases in said utility.

Q-learning with stochastic enforcement

The results from Q-learning with stochastic enforcement (see Figure 3.3) show a similar graph, where the agent (albeit slightly) prefers to not check in when there is no (or very little) enforcement. When the enforcement intensity is 0.05 or higher, the prefered policy switches to checking in, which reaches a constant value for average utility. The average utility for not checking in decreases as the enforcement intensity increases, similar to the experiment with fixed enforcement. The average utility for checking in stays constant shortly after the critical value for enforcement intensity is reached. Further research need to be conducted to explain why further increases of the enforcement intensity have no effect on the average utility for checking in when enforcement is stochastic, while it does have effect when enforcement is fixed.

General discussion and future research

The original idea for this thesis was to use reinforcement learning on both parties in the public transport problem. The traveling agent could learn policies based on the enforcement strategies of the normative organization, while the normative organization could learn policies for employing enforcement based on behavior of the traveling agent. This was suggested as future research by Li *et al.*[10]. I wanted to build up to that by giving sufficient background information and theory, and starting with just a learning traveler as regards the programming section. During the process it became clear that the learning normative organization fell out of the scope of this

Bachelor's thesis. I think it would be a very interesting subject for future research, which could give more insight in effective enforcement strategies. This could potentially be coupled with Game Theory, a topic I considered including in this thesis as well. The public transport problem consists of two parties picking strategies based on the opponent's action and a reward table, so Game Theory could provide useful insight on resulting strategies and possible equilibria. I investigated a variant of Game Theory called Stackelberg Security Games which I initially planned to cover in this thesis, but unfortunately it did not fit in the scope of this thesis to include a sufficient analysis.

A potential flaw in this project could be the decision to use Q-learning on the public transport problem. It is the RL algorithm I am most experienced and confident with, and it is commonly used for RL. Additionally, it was used by Li *et al.* from whom I have drawn inspiration for this project. However, the one-dimensional model could not be exploited optimally by Q-learning, as there is always a maximum of one state to travel to from any state. The maximal expected value for s' is not relevant when deciding on an action in state s when the agent ends up in the same state regardless of the action chosen. I considered implementing a different RL algorithm in addition to Q-learning, which was a Policy Gradient algorithm used for the Cartpole problem.[8] In the Cartpole problem the agent must learn to balance a vertical pole on a cart by pushing the cart left or right. The idea of Policy Gradient is to initialize a certain policy at the start, which will be updated iteratively based on improvement in expected payoff for the agent. The algorithm does this by using a set of parameters θ and a value function. The parameters θ are initialized randomly and are updated iteratively to maximize this value function, which is often just the total sum of rewards.[3] As the agent in the Cartpole problem works with a binary action space like in my public transport problem, I thought the algorithm could be translated for and applied to my model. After implementing the Policy Gradient algorithm it did not seem as though the agent was able to learn based on the rewards. Even after setting the reward for defying the norm undetected a thousand times higher than the reward for complying, the agent's policy still converged to always checking in often. This could be because the Cartpole problem worked with a very simple reward system (a reward of +1 for every step in which the pole is kept upright) whereas mine is more complex with two different high rewards and one lower reward. Delving deeper into the Policy Gradient fell out of the scope of this project, which is why I decided to focus strictly on Q-learning. It is an interesting idea for future work to implement different methods of reinforcement learning on the public transport problem nonetheless.

Conclusion

This thesis shows it is possible to discover policies of agents in a simulation based on the normative situation that is public transport. In turn, this allows normative organizations to adapt their strategy of enforcement, to catch as many norm defyers with the limited resources they have available to them. This answers my research question: "How can enforcement strategies for public transport be optimized by making use of learning agents?" The results of this thesis could be used by normative organizations for insight in behavior of agents in normative situations, while the approach I used can be built on to conduct further, more complex research on behavior in normative situations.

Bibliography

[1] N. Alechina, M. Dastani, and B. Logan. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1057–1064, 2012.

[2] N. Alechina, M. Dastani, and B. Logan. Norm approximation for imperfect monitors. 2014.

[3] B. Banerjee and J. Peng. Adaptive policy gradient in multiagent learning. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 686–692, 2003.

[4] M. B.V. Monitor zwartrijden stads- en streekvervoer - resultaten meting 2006 - eindrapport, 9 2006.

[5] Z. Cankara. Q-learning algorithm for the cliff walking problem. https://github.com/zeynepCankara/Cliff-Walking-Solution/. Accessed: 11-05-2020.

[6] Geweld tegen ns-personeel vooral door zwartrijders. https://beveiligingnieuws.nl/nieuws/geweld/geweld-ns-personeel-vooral-zwartrijders/. Accessed: 11-06-2020.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[8] J. Klaise. Reinforcement learning with policy gradients in pure python. https://www.janisklaise.com/post/rl-policy-gradients/. Accessed: 01-06-2020.

[9] J. Li. Development of intelligent agents for playing angry birds game. May 2014.

[10] J. Li, F. Meneguzzi, M. Fagundes, and B. Logan. Reinforcement learning of normative monitoring intensities. In *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pages 209–223. Springer, 2015.

[11] F. Meneguzzi, B. Logan, and M. Silva Fagundes. Norm monitoring with asymmetric information. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1523–1524. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[12] P. Norvig and S. Russel. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.

[13] J.-C. Pomerol. Artificial intelligence and human decision making. *European Journal of Operational Research*, 99(1):3–25, 1997.

[14] M. Silva Fagundes et al. Sequential decision making in normative environments. 2012.

Appendix A

Python Q-learning Implementation

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 11 12:00:18 2020
Based on a Q-learning algorithm used by Zeynep Cankara for the Cliff-Walking problem: https://github.com

@author: Rick Mank
"""

import numpy as np
import matplotlib.pyplot as plt


def create_Qtable (rows = 1, cols = 7):

    #initialize table with Q-values for both actions
    q_table = np.zeros((2 , rows * cols))
    return q_table


def epsilon_greedy_policy(state, q_table, epsilon = 0.1):

    # choose a random int from an uniform distribution [0.0, 1.0)
    decide_explore_exploit  = np.random.random()

    #compare random int with epsilon to decide whether to explore or to exploit
    if(decide_explore_exploit < epsilon):
        action = np.random.choice(2) #exploration (NOT-CHECK-IN = 0, CHECK-IN = 1)
    else:
        action = np.argmax(q_table[:, state]) #exploitation: choose the action with largest Q-value (sta

    return action


def get_state(agent, q_table):

    # get position of the agent
    (posX , posY) = agent

    # obtain the state value
    state = 7 * posX + posY

    # get maximum state value from the table
    state_action = q_table[:, int(state)]
    maximum_state_value = np.amax(state_action) # return the state value for the highest valued action

    return state, maximum_state_value



def enforcement(enf_intensity):

    #loop over numpy array, placing 1s when random int is lower than the enforcement intensity
    enf_map = np.zeros(7)
    for day in range(0, len(enf_map)):
        enforce_or_not = np.random.random()
```

```
        enf_map[day] = 0

        if enforce_or_not < enf_intensity:
            enf_map[day] = 1

    #return array of 0s and 1s, where a 0 means no enforcement on that day, and a 1 means enforcement
    return enf_map


def stoch_enforcement(enf_intensity):

    #creates enforcement map to use for stochastic enforcement, which changes for every episode
    stoch_map = []
    for i in range(7):
        choice = np.random.choice([0, 1], p=[1-enf_intensity, enf_intensity])
        stoch_map.append(choice)

    return stoch_map


def get_reward(state, action, enf_map):

    #get reward based on action chosen, and value of the day(state) in the enforcement map
    if action == 0:
        if enf_map[state] == 1:
            reward = 50
        else:
            reward = 100
    if action == 1:
        reward = 95

    return reward


def move_right(agent):

    #increase Y coordinate by 1 to signify the next day
    listified = list(agent)
    if(listified[1] < 6):
        listified[1] += 1
    agent = tuple(listified)

    return agent


def checked_in(agent, action, env):

    #change current value in environment to 1 if the agent decided to check in
    (posY, posX) = agent
    env[posY][posX] = 0
    if action == 1:
        env[posY][posX] = 1
    return env

def update_qTable(q_table, state, action, reward, next_state_value, gamma_discount = 0.9, alpha = 0.5):

    #update the q_table based on observed rewards and maximum next state value via
    #Q(S, A) <- Q(S, A) + [alpha * (reward + (gamma * maxValue(Q(S', A'))) -
Q(S, A)
    update_q_value = q_table[action, state] + alpha * (reward + (gamma_discount * next_state_value) - q_
    q_table[action, state] = update_q_value

    return q_table


def qlearning(counter, enf_int, num_episodes = 100000, gamma_discount = 0.9, alpha = 0.5, epsilon = 0.1)

    #arrays for calculating means
    tenmeanzero = []
    tenmeanone = []
    for i in range(10):
        q_table = create_Qtable()
```

```
        meanvalues_zero = []
        meanvalues_one = []
        agent = (0, 0) # starting from left
        enf_map = enforcement(enf_int) #un-comment for fixed enforcement

        # start iterating through the episodes
        for episode in range(0, num_episodes):
            #enf_map = stoch_enforcement(enf_int) #un-comment for stochastic enforcement
            env = np.zeros((1, 7))
            agent = (0, 0) # starting from left
            step_cum = 0
            while(step_cum < 7):
                #get the state from agent's position
                state, _ = get_state(agent, q_table)
                # choose action using epsilon-greedy policy
                action = epsilon_greedy_policy(state, q_table)
                #keep track when agent checks in
                env = checked_in(agent, action, env)

                # observe next state value
                next_state, max_next_state_value = get_state(agent, q_table)
                # observe reward
                reward = get_reward(next_state, action, enf_map)

                # update q_table
                q_table = update_qTable(q_table, state, action, reward, max_next_state_value, gamma_disc
                #move to next day
                agent = move_right(agent)
                # update the state
                state = next_state
                step_cum += 1
            meanvalues_zero.append(np.mean(q_table[0]))
            meanvalues_one.append(np.mean(q_table[1]))

            #to keep track during run
            if(episode > 99998):
                print("Episode ", i + 10*counter)
        tenmeanzero.append(np.mean(meanvalues_zero))
        tenmeanone.append(np.mean(meanvalues_one))
    #return average Q-values for both actions
    return np.mean(tenmeanzero), np.mean(tenmeanone)


enf_array = []
meanzero_array = []
meanone_array = []
for i in range(11):
    enf_array.append(i/10)
    meanzero, meanone = qlearning(i, i/10)
    meanzero_array.append(meanzero)
    meanone_array.append(meanone)
print(meanzero_array, meanone_array)
plt.plot(enf_array, meanzero_array, label = "Not checking in")
plt.plot(enf_array, meanone_array, label = "Checking in")
plt.xlabel('Enforcement intensity')
plt.ylabel('Average utility')
plt.xticks([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
plt.legend()
plt.show()
```