ARTIFICIAL INTELLIGENCE: FINAL PROJECT ASSIGNMENT

An AI-powered solution to control the traffic flux in an intersection.

Jorge Lázaro Ruiz (100452172)
Laura Belizón Merchán (100452273)

Table of contents

1. Executive summary

1.1. The problem

This project is designed to **efficiently control the traffic flux in an intersection by operating three traffic lights.** Using sensors in each of the streets that conform the intersection – west, north and east – we can detect if the traffic flux level on each of them is either high or low, giving us a total of eight different possible states. Through an analysis of historical data gathered for roughly 48 hours, we deduce that turning on and off each of the green lights affects the traffic flux of each road, so it follows that **there must be an optimal choice for which light to turn green on each case.**

1.2. The solution

These optimal choices are what we strived, and managed, to find through modelling an automaton to control this problem with a **Markov Decision Process** (MDP). This MDP receives the probability of each transition via transition tables obtained by previously processed historical data and **tells the automaton which action to take given one of the eight states**, namely, the optimal policy for it.

1.3. The value and importance of our solution

Our automaton provides a solution that a human is unable to. When deciding on which traffic light to turn green, a human may be tempted to simply turn on the most crowded direction's corresponding green light, but this isn't always the best choice – as proved by the historical data, there is even a chance that turning a green light on may increase the traffic flux of that direction rather than decrease it.

Artificial Intelligence (AI) is undoubtedly the best approach to this problem, free from human error, learning from data gathered in the past and therefore always making the optimal decision.

2. Objectives

Our goal when developing this project was to **find the best course of action** (optimal policy) given the state of the traffic flux and have our controller apply it until we reach a low traffic level on every direction of the intersection (our goal state).

The objective was also to make this solution **easily adjustable** to any other intersection, not just this one, which is why we converted each state and action to a number and defined modifiable constants for every number particular to our problem.

3. Formal description of the AI model

An MDP consists of four basic elements: states, actions, transitions and rewards.

3.1. States

We have three different directions: west, north and east. Each of these directions can be in a state of high or low traffic flux. This leaves us with a total of 2^3 possible states:

Low, Low, Low	Low, Low, High	Low, High, Low	Low, High, High
High, Low, Low	High, Low, High	High, High, Low	High, High, High

Table 1 All possible states of the automaton

In *Table 1*, and for the rest of this report, the first traffic level corresponds to the west road, the second one to the north road and the third one to the east road.

State *Low, Low, Low* constitutes our goal state, that is, the state we wish to be in and where our AI will stop its execution.

3.2. Actions

The MDP may take one of three actions:

Turn on the **west** green light	Turn on the **north** green light	Turn on the **east** green light

Table 2 All possible actions for the AI

3.3. Transitions

A priori, it may seem like we have a total of $8 \times 3 \times 8 = 192$ possible transitions between the states. However, some of these transitions do not appear on our historical data and therefore have probability 0 of happening. These are impossible transitions, such as starting at *Low, Low, Low* and going to any other state. Eliminating these 123 impossible transitions, we end up with a total of 69 possible transitions.

The probability of every transition is easily accessible by just using the data stored in the 3-dimensional array *T*. Its usage is explained in the source code.

3.4. Rewards

For the reward function, we will use the cost of each action. Since we gather the data every 20 seconds, we consider the cost to be equal to the amount of time we wait (in seconds) until we receive a new state and the MDP can make a choice again.

Cost of WEST	**Cost of NORTH**	**Cost of EAST**
20 seconds	20 seconds	20 seconds

Table 3 Cost of each action

3.5. Graphical representation

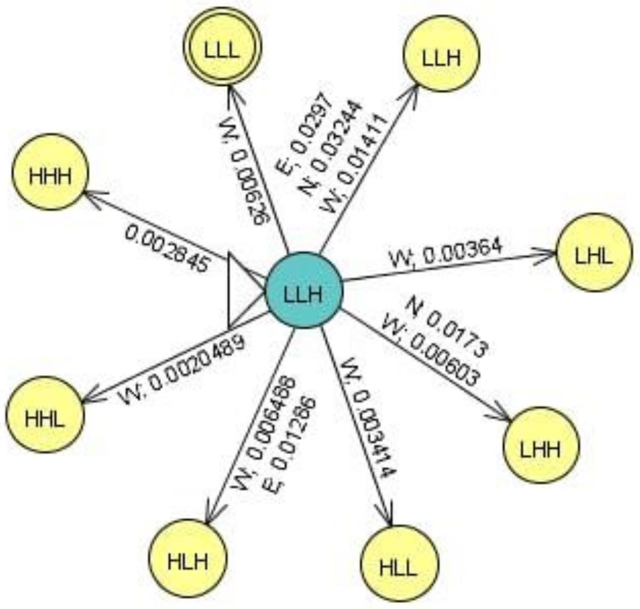

Image 1 Graphical representation of the choices the AI can make from the Low, Low, High state using JFLAP.

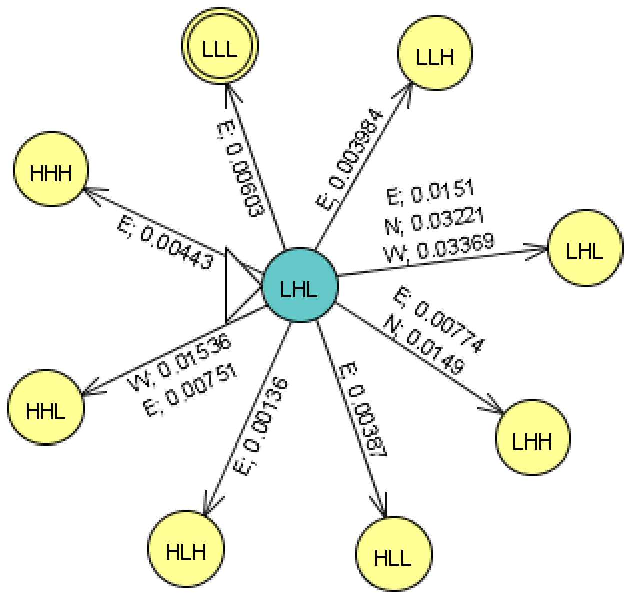

Image 2 Graphical representation of the choices the AI can make from the Low, High, Low state using JFLAP.

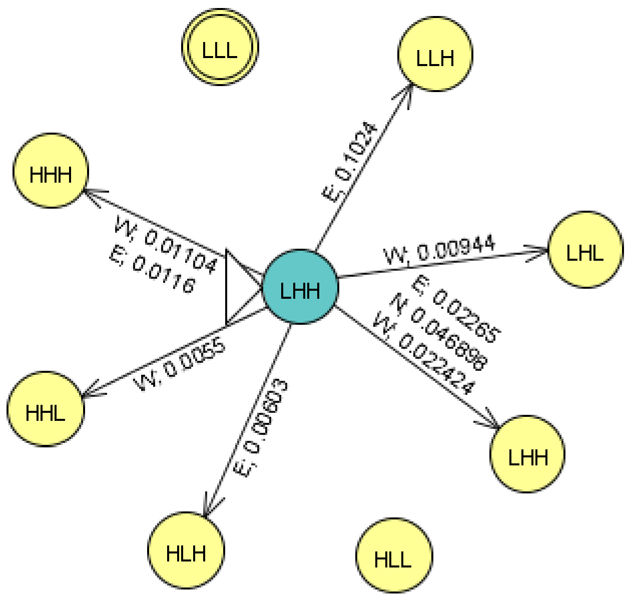

Image 3 Graphical representation of the choices the AI can make from the Low, High, High state using JFLAP.

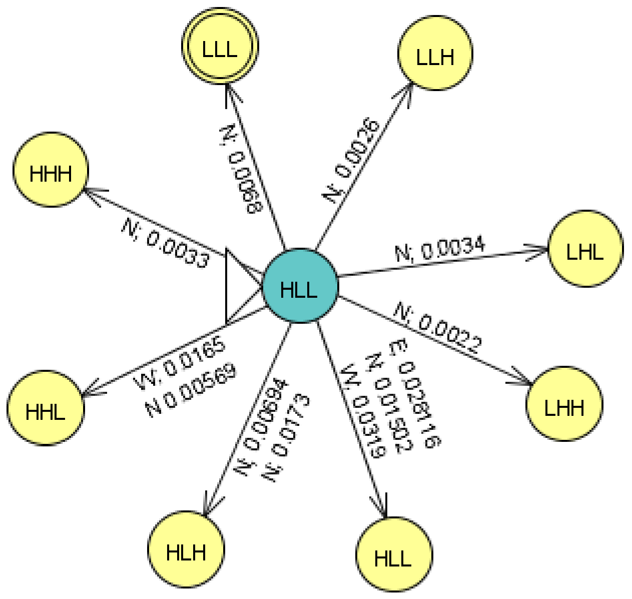

Image 4 Graphical representation of the choices the AI can make from the High, Low, Low state using JFLAP.

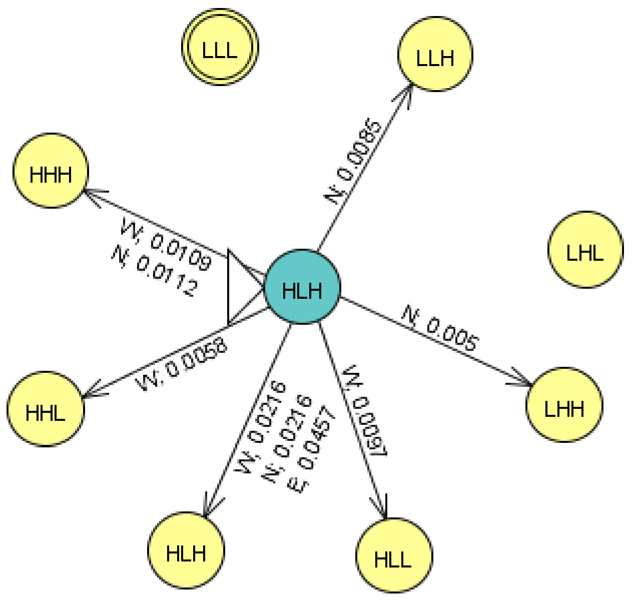

Image 5 Graphical representation of the choices the AI can make from the High, Low, High state using JFLAP.

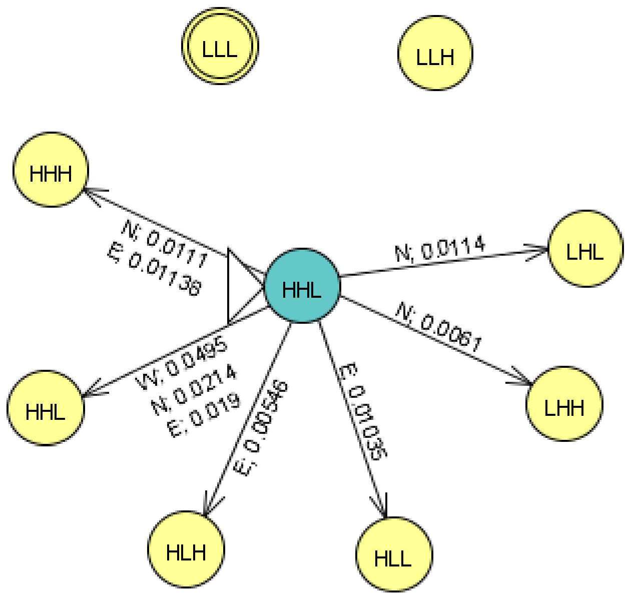

Image 6 Graphical representation of the choices the AI can make from the High, High, Low state using JFLAP.

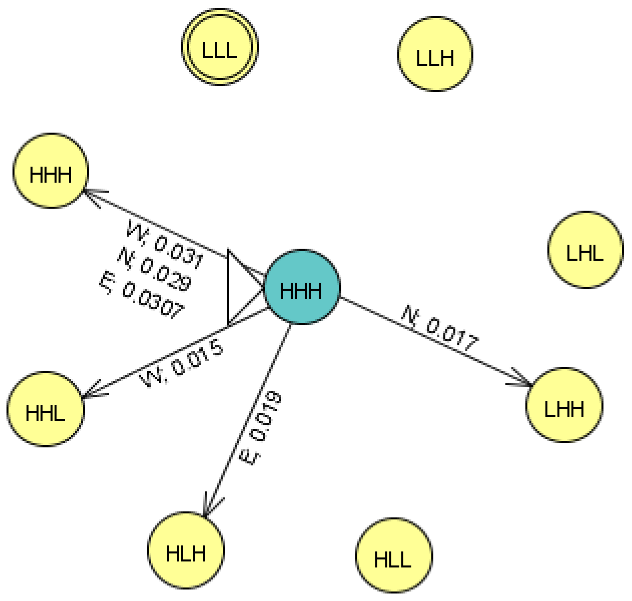

Image 7 Graphical representation of the choices the AI can make from the High, High, High state using JFLAP.

Due to the amount of states and transitions, we have divided the MDP into seven different diagrams to illustrate the transitions from any given state rather than all together so that the graphical representation is easier to understand.

4. Methodology

In order to design this MDP and find the optimal policies, we followed the usual steps:

1. Analyze the problem and identify which states, actions, transitions and rewards are needed.
2. Define each of these elements.
3. Find the Bellman equations in order to update the expected value of each of the states.
4. Perform a value iteration to determine the convergent expected value of every state.
5. Identify which policy is optimal for each of them.

The dimensions of this problem were too big to design this automaton manually, so we used a Python program to help us manage this data. The development of this code is detailed in its own section.

5. Resources used

We chose to use Python to write the code that assisted us in designing the MDP because of its easy portability and versatility. Additionally, we used the library *difflib* to make our automaton immune to human errors when transcribing the historical data fed to it.

In order to graphically represent the MDP as an automaton, we used the free Java-based software *JFLAP*.

On a more theoretical note, we used concepts from the book *Artificial Intelligence: A Modern Approach* by Stuart J. Russell and Peter Norvig.

6. Development

To start with, we had to **analyze the problem** to determine the way we wanted to approach and model it. We quickly identified the historical data provided to us as a way to determine which transitions happened on each state and with which frequency, so we had found the source for our probability table.

In order to **extract probabilities from the historical data**, we coded an algorithm that reads each of the lines, parses them in a *state-action-new state* scheme, translates this into a three-digit code identifying them and adds one more instance of this transition to a **3-dimensional array where we will store the probability** (assigned using the rule of equally likely events). The code translation is more than direct string comparison, as we assumed that some problems may arise with a stiff method like that and the algorithm would collapse due to typographical errors, data corruption or invisible characters appearing when converting data, so we used the function *difflib.get_close_matches()* for a **more refined and flexible way of identifying each state** with its numerical code.

Once the probabilities are found, we **found the Bellman equations in order to find the expected value** of each of the states. To do this, we developed the **function *bellman()*,** which takes the current state and previous values as arguments (and the possible states and actions to generalize the problem, as stated in Section 2) and outputs the next expected value of the given current state, with a precision of six decimal places.

In order to **find the optimal policy for each of the states, we had to perform the value iteration algorithm** until the values converged, i.e., the last two iterations were identical. This gave us the expected values of all eight states. When plugging these into the Bellman equations, we can find the action that gives us the minimal expected value on each of them, and that action is, by definition, the optimal policy for that state.

7. Results

7.1. Question 1

The input data does not include any cases where the starting situation was low traffic level in all three directions. Is this normal? If you had this kind of data, what would happen or what should we have done?

If the starting situation is *Low* in all three directions, we already reached our goal state. It is logical that no data exists where this was the starting situation. Our AI stops execution when this happens.

Had we encountered this kind of data, we would have had to analyze these transitions to obtain which states are the most likely to restart the AI after the traffic is stabilized. Also, we would have to note the most common action among the transitions starting on *LLL* as the worst possible action to take after the traffic is stabilized.

7.2. Question 2

The statement does not say anything about the cost of the actions. What reasonable assumption could we make?

As specified in the section on rewards in Section 3.4, we have assumed the cost is uniform, as turning on any of the green lights has equal cost. We have therefore considered the cost to be the time that we are committed to each of the decisions that our MDP makes at any given time, 20 seconds.

7.3. Question 3

What are the expected values of the states? Provide the values to six decimal places. The precision must be greater than one thousandth.

The values obtained and stored in the array *value_iterations* are shown in *Table 4*, where the rightmost column is the final expected value of each state. Six to seven iterations were needed for this value to converge, depending on the state, except for the goal state *LLL* which, by definition, has an expected value of 0 on all iterations.

	0	*1*	*2*	*3*	*4*	*5*	*6*	***7, 8, …, ∞***
LLL	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
LLH	0	20	20.771770	20.804715	20.806184	20.806250	20.806253	**20.806254**
LHL	0	20	20.881047	20.920119	20.921875	20.921955	20.921958	**20.921959**
LHH	0	20	20.937962	20.981951	20.984014	20.984111	20.984115	**20.984116**
HLL	0	20	20.785430	20.819108	20.820604	20.820672	**20.820675**	**20.820675**
HLH	0	20	20.915196	20.957076	20.958992	20.959080	**20.959084**	**20.959084**
HHL	0	20	20.924303	20.965687	20.967550	20.967634	**20.967638**	**20.967638**
HHH	0	20	20.937962	20.981746	20.983763	20.983855	**20.983860**	**20.983860**

Table 4 Values obtained and stored in the array value_iterations

7.4. Question 4

What are the expected values of the states? Provide the values to six decimal places. The precision must be greater than one thousandth.

The optimal policy obtained for each of the eight possible states in this problem as stored in the dictionary *{op}* are shown in *Table 5*.

State	*LLL*	*LLH*	*LHL*	*LHH*	*HLL*	*HLH*	*HHL*	*HHH*
Optimal policy	Undefined	West	East	North	North	East	East	West

Table 5 Optimal policy for each state

The optimal policy for the goal state is undefined, as we do not need to take any actions once we arrive to it.

7.5. Question 5

If we had also had incoming traffic from the South and for each direction we had measured traffic at 5 levels instead of 2, what would have changed in the problem?

For starters, we would have had 5^4 = 624 different possible states, which would have made the problem much more complicated but also our solution more refined thanks to the more detailed information that those five traffic levels would have given us.

The introduction of three more traffic levels (hypothetically: *Low*, *Low-Mid*, *Mid*, *Mid-High*, *High*) could also help us adjust what we consider to be a "good" level of traffic. We could determine that *Low* and *Low-Mid* are both acceptable traffic levels or we could be stricter and only accept *Low*.

Additionally, assuming the involvement of a fourth direction, south, would also give us a fourth traffic light, this would give us a new action to take: turning on the southern green light, and a total of $624 \times 4 \times 624 = 1{,}557{,}504$ different transitions a priori.

8. Conclusions

This project proved to be a very interesting look on how artificial intelligence can be applied to optimize problems in our daily life. We had a lot of freedom to find a solution, which was **both an engaging and a daunting aspect.** Engaging because we had the opportunity to put our abilities and knowledge of artificial intelligence development to the test, but daunting because this was the first time we had to apply it to coding.

We didn't have any **experience working with CSV files**, so we had to research them and develop a way of processing the historical data. We considered importing the *csv* library, but ultimately decided to code our own algorithm so that we could have more control over it and adjust it perfectly to the type of problem we were facing.

The **amount of data** was also much bigger than what we were used to working with, as were the number of states and, consequentially, the number of transitions, but after a lot of testing and rewriting the code, we learned to manage them.

Initially, we planned to manually identify each of the transitions, but after realizing that there were almost two hundred of them, we realized we had to **find a way to reduce the number** of *if* statements used to identify them. That is when we came up with the code system we ended up using, which restructures the identification problem into three separate identifications rather than a combination of all three. This gave us $8 + 3 + 8 = 19$ possible identifications, less than a tenth of the original plan, and fit nicely with the 3-D matrix we intended to use as a probability table.

After that, we simply implemented ways to apply the methods we use to design MDPs and find optimal policies. The main difficulty on this step was our **lack of experience with Python,** accentuated by the fact this was the first time we had to code in order to this kind of problem. We researched multidimensional arrays, the use of *continue*, library usage, file management and string manipulation, all concepts we already knew from working with MATLAB or C/C++, but not Python, which we have been learning this same semester.

9. Budget

9.1. Staff

Project managers	2
Software developers	2
Testers	2

Table 6 A detailed breakdown of the team that developed this project

Role	Workdays	Salary per workday	Total cost
Project manager	2	€500	€1000
Software developers	8	€400	€3200
Testers	2	€300	€600
			Total budget: €4800

Table 7 Breakdown of the total budget of the project