Final Project Assignment

Markov Decision Process For Traffic Control

Artificial Intelligence

Raúl Manzanero López-Aguado - 100451106

Elena Esther Pajares Palomo - 100451238

Group 88

Bachelor in Computer Science & Engineering

Contents

Executive Summary

The objective of this project is to reach a low level of traffic in every direction of a traffic intersection regulated by traffic lights. For this purpose we obtained the optimal policy of a Markov Decision Process that is designed at an initial point with eight different states ($HHH, HHL, HLH, HLL, LHH, LHL, LLH, LLL$) and three actions (N, E, W), as well as the transition function and probability tables, derived from the given data, and a cost function.

To get the optimal policy we developed a code that computed the expected values by using Bellman Equation and the optimal policy for each state with an associated cost per action of 20, based on the time that each cycle lasts.

This code has been designed in Python trying to be as generic as possible so it can adapt to different sets of states and actions and their costs, and it has been properly tested.

The overall cost of the project has been calculated to be around 2500€.

Objectives

The main objective of this project is to design an automaton that opens the appropriate traffic light at each cycle following the optimal policy corresponding to a Markov Decision Process (MDP) for reaching a low level of traffic in all three directions of the intersection, given that out of the three, only one light in the street can be turned green in each cycle. This problem has to be solved in the most generic way possible.

For the purpose of solving this problem, we will have to calculate all the conditional probabilities based on the data given, compute the expected values using Bellman Equation for all states and finally calculate the optimal policy of each of the states.

Formal description of the AI model

Markov Decision Process:

$$MDP = < S, A, T, C >$$

Set of states:

$$S = \{HHH, HHL, HLH, HLL, LHH, LHL, LLH, LLL\}$$

The set of states is composed of states ‘s’ with three-character strings. Each character of the string represents the traffic level of each street in the order North, East, West, so that H corresponds to high and L to low. We have computed all different combinations to obtain all possible states.

As the objective is to decrease the traffic level in all directions, the goal state will be LLL.

Set of actions:

$$A = \{N, E, W\}$$

Each of the actions of the set represent the direction in which the light is turned green in each cycle, being N North, E East, and W West.

Transition function:

$$T = T(s, a, s') = P(S_{t+20} = s' | A_t = a, S_t = s)^1$$

We will see that each transition has a probability based on the historical data provided (data.csv) that we will use to calculate the probabilities in further sections to calculate the expected values using Bellman Equation as well as the optimal policies. Note that this represents the instant of the time in which we are.

	P(S|N,HHH)	**P(S|N,HHL)**	**P(S|N,HLH)**	**P(S|N, HLL)**	**P(S|N, LHH)**	**P(S|N, LHL)**	**P(S|N, LLH)**	**P(S|N, LLL)**
HHH	0.6282	0.2222	0.2420	0.0716	0	0	0	0
HHL	0	0.4263	0	0.1234	0	0	0	0
HLH	0	0	0.4645	0.1506	0	0	0	0
HLL	0	0	0	0.3259	1	0	0	0
LHH	0.3717	0.1224	0.1100	0.0494	0	0.3164	0.3478	0
LHL	0	0.2290	0	0.0741	0	0.6836	0	0
LLH	0	0	0.1834	0.0568	0	0	0.6522	0
LLL	0	0	0	0.1481	0	0	0	0

	P(S|E,HHH)	**P(S|E,HHL)**	**P(S|E, HLH)**	**P(S|E, HLL)**	**P(S|E, LHH)**	**P(S|E, LHL)**	**P(S|E, LLH)**	**P(S|E, LLL)**
HHH	0.6164	0.2463	0	0	0.2297	0.0886	0	0
HHL	0	0.4113	0	0	0	0.1500	0	0
HLH	0.3836	0.1182	1	0.3809	0.1194	0.0273	0.3021	0
HLL	0	0.2241	0	0.6190	0	0.0773	0	0
LHH	0	0	0	0	0.4482	0.1545	0	0
LHL	0	0	0	0	0	0.3023	0	0

[1] We have considered S_t as the state when time t is 0s and S_{t+20} when time t is 20s, since each cycle when an action is performed lasts a total of 20 seconds.

LLH	0	0	0	0	0.2027	0.0795	0.6978	0
LLL	0	0	0	0	0	0.1204	0	0

	P(S|W, HHH)	**P(S|W,HHL)**	**P(S|W, HLH)**	**P(S|W, HLL)**	**P(S|W, LHH)**	**P(S|W, LHL)**	**P(S|W, LLH)**	**P(S|W, LLL)**
HHH	0.6796	0	0.2269	0	0.2277	0	0.0635	0
HHL	0.3203	1	0.1206	0.3404	0.1150	0.3132	0.0457	0
HLH	0	0	0.4492	0	0	0	0.1447	0
HLL	0	0	0.2033	0.6596	0	0	0.0761	0
LHH	0	0	0	0	0.4624	0	0.1345	0
LHL	0	0	0	0	0.1948	0.6868	0.0812	0
LLH	0	0	0	0	0	0	0.3147	0
LLL	0	0	0	0	0	0	0.1396	0

Cost function:

C = C(s, a)

In this specific case, we will use a cost function instead of a reward function since we have considered that opening a semaphore has an associated cost. That is why we have chosen that we have a time cost consisting of 20 seconds, since in the initial state we are at time t = 0s and after the action is fully performed and the new state is obtained, we are in time t = 20s.

As we will see, the cost model we have is stochastic, since an action has a probabilistic effect.

Methodology

For the development of the project, we have followed a series of steps just to have each functionality working on time and to have a good administration of our time before the delivery.

We started by fully understanding the tasks and questions that we were asked of. Then, we broke down the problem into a series of steps: 1. Getting the probabilities, 2. Getting the Bellman equations, 3. Computing the optimal policy, and then completing the report and the video.

For the development of the code, we initially wrote down some general pseudocode in which we divided the different pieces and functions and worked down how they will be interacting together in the final version. Then, we started to program the code following the order mentioned above and tested each function separately as we completed them to try to make sure that no mistakes were being dragged into the others. In this process, we found some mistakes in our original design and algorithms that we solved right away. The development of the code for the different functions was originally divided between the two members of the group but ultimately, we found ourselves working together, talking through the mistakes and why some things were not giving the expected result, and solving each other's doubts and errors.

Finally, we completed the media pieces required, such as the report and the video.

Resources used

During the development of the project, Python language was used. As a tool to help us code simultaneously we used the web Replit, and uploaded the code to a private repository on GitHub in order to keep the different versions as the project progressed in case any mistakes were made when implementing new functions. We also had some issues while calculating some of the data, so we resorted to looking for Markov Decision Process algorithms in some websites, but in general, it was enough with the knowledge we had from previous years.

Development

To start with the development of the project, we have considered three different states. Firstly, a design stage in which we defined what to do and how to fully understand the concept of the project and knowing what were the initial states, the possible operations and the expected results by looking at the final goal we needed to achieve. Afterwards, the implementation phase, which we consider the most important of the project to exactly know what is happening with our data, was performed. In this stage, we needed development of software for performing the operations we wanted, since dealing with big amounts of data can be difficult to handle if we perform operations by hand, so we decided to follow a dynamic programming approach to make the task easier. Finally, after the code was implemented and results were obtained as asked, we needed to test that the results obtained matched the expected ones. Since we had no way of checking that, some tests with a smaller database were done.

Design

For the design, we have decided to divide the project into four different files, each corresponding to a step of the process and containing the necessary functions.

These files are: *probability_calculation.py,* that contains the function CalculateProbabilities to get all needed probabilities with the data given, *bellman_equation.py*, that contains a main function called Bellman Equations that utilizes two auxiliary functions, bellman and summatory to help in making the code more readable and returns the expected values for every state, and *optimal_policies.py* which contains a main function OptimalPolicies that uses an auxiliary function summary to obtain the optimal policy for each state. Finally, the *main.py* file declares the needed variables (treated as constants during execution) such as states, actions, cost, etc and calls the functions from the other files.

The design has taken into consideration readability and maintainability, by having a clean and commented code, separated in specific files, and it is also pretty general, as it could easily be changed to accommodate different state and action names and a bigger or smaller amount of them.

Implementation

For the implementation of the functionality of the processes that must be carried out, we used a dynamic programming approach by the extensive use of Python functions, lists and dictionaries. No libraries were used apart from os library to deal with the file path of the data file and csv library to deal with the opening and reading of information from the csv file with all the data provided.

Since we wanted a generic program that can be used in other similar situations, we decided to use constants in the main program just to change them if needed without having to change all the code. Those constants are:

- **STATES** → States are not the possible states of the plot, but the elements that form the states, which are H (High) and L (Low) in our initial design. After this, all possible states of length 3 will be generated. This list can contain N different elements.
- **ACTIONS** → Possible actions that can be carried out. In our design, we have E (East), W (West), and N (North), which are the actions of putting a light of a semaphore in green. This list can contain N different elements.
- **GOAL_STATES** → It is a simple list with the goal states that we want to consider, being strings composed of items from the constant STATE and with a compulsory length of 3 for this project.
- **COSTS** → Costs are stored in a dictionary whose keys are the different actions (N, E, W) and the values of each key are the associated costs, which are 20 all of them for our design.
- **FILE** → This is the name of the file that we want to use to test or to obtain information. The complete path of the file inside the respective functions.

Just giving the information explained above to the main program in a correct way, we can obtain the transition function of probabilities, the expected values by using Bellman Equation and the optimal policies for each state. This is done in our code in a modular way, by the implementation of four Python files with different functionalities that are called between them.

probability_calculation.py

This file contains a function, *CalculateProbabilities*, which receives the state's list, the actions to be performed and the file with the data, and returns a dictionary with the probabilities calculated. In this dictionary, each key is a probability with the form **S'|a,S** which is stored in this way for the sake of simplicity in further functions. Each key contains a number with the probability of each transition based on the readings of the file provided as parameters.

To perform the operations of probabilities calculations, we have used several dictionaries to carry out the necessary operations, as each row probabilities must sum one. Furthermore, it can be seen in the code that there are commented prints that would help the user to see what is happening after each operation by just using them, being a useful tool for us while debugging during the implementation of the functions.

bellman_equation.py

The bellman_equation.py contains the main function, BellmanEquation, which receives the list of states, goal state, actions, and costs to compute and return a dictionary with the values for each state. This function calls an auxiliary one called *bellman* to compute the equations for each individual state and return the minimum out of all. Bellman also uses another auxiliary function called *summation*, which computes the summation inside the Bellman equation. These decisions have been made in order to make the code more readable. The function from calculate_probabilities.py is used inside to get all of them and use them. This file contains, in fact, the Bellman Equation we saw in class but fully implemented in code.

optimal_policy.py

The optimal_policy.py contains the main function called *OptimalPolicy* which calculates the optimal policy for each state and returns a dictionary with all of them. This function is also called an auxiliary one to calculate the summation. Note that we have not included the goal states (LLL in our case) in the final result.

main.py

Finally, as explained at the beginning of the section, the main file is where the user selects the sets to use. Moreover, we have included a call to *BellmanEquation* and *OptimalPolicy* functions to answer the questions that we will see in the next point.

Testing

For testing purposes, we have used a new database (*tests.csv*) which contains less information than the original provided (*data.csv*). We decided to do this in order to perform some Markov iterations by hand and check if our code was working as expected with the complete set of data, since we have not found any way of checking if both the probabilities, expected values and optimal policies where accurate by using the full set of information provided which contains more than 8000 lines of historical data.

Results

Q1 - Is it normal to not have any cases in the input data with low traffic in every direction?

As low traffic in every direction is the goal state, it is obvious that we do not need any data regarding it. However, if that information was provided, it will not matter either, as LLL, because it is the goal state, and therefore, all probabilities associated with it in the summation (summation) of the Bellman equation will be multiplied by zero and the result will stay the same. Otherwise, if the probabilities with LLL are provided and they are not 0, the results could be inconsistent with the problem definition, since we know that the goal state should be LLL and it has no sense to do an action in the automaton while being in the goal state.

Q2 - Cost of action

As the cost was not given, we have decided to assign a cost of 20 to each action. The cost of all actions is the same because we do not need to prioritize one over the other, as we just want to reach the final state as much as possible, no matter the order in which we do it. We have setted it to 20 considering the cost as time cost, as this is the amount of seconds that each cycle lasts. Moreover, since the three costs are the same, we could also have considered the cost of unitary (cost 1 for all of them) and the results of the optimal policies would have been the same. Therefore, we have that the cost of the actions are $C(a) = 20 \ \forall \ a \in A$.

Q3 - Expected value states

After computing the Bellman equation, we obtain the following expected values for each state. Note that V(LLL) must be 0 since it is a goal state.

V(HHH) = 794.272892

V(HHL) = 739.009122

V(HLH) = 742.130036

V(HLL) = 587.402741

V(LHH) = 745.360440

V(LHL) = 615.223709

V(LLH) = 593.162010

V(LLL) = 0

Q4 - Optimal policy

After computing the optimal policy, we obtain the following result for each state. The optimal policy for the goal state is not taken into account as it already is the state that we want to reach.

$$\pi^*(HHH) = E$$

$$\pi^*(HHL) = E$$

$$\pi^*(HLH) = W$$

$$\pi^*(HLL) = N$$

$$\pi^*(LHH) = E$$

$$\pi^*(LHL) = E$$

$$\pi^*(LLH) = W$$

Q5 - Problem changes for more directions and levels

If we add more states and traffic values to the problem, it will become more complex adding one more action to the set of actions and severely increasing the number of states from eight to 5^4. In terms of the resolution, we could still use the same code for the computation after doing minimal changes.

Markov Decision Process:

$$MDP = < S, A, T, C >$$

Set of states:

$$S = \{p^4;\ p \in H, M, L, V, B\}$$

The set of states will be made out of 5^4 states with a combination of the five states that could be such as H, M, L, V, B, where, for instance, H is High, M is Medium, L is Low, V is Very Low and B is Very High.

Set of actions:

$$A = \{N, E, W, S\}$$

Transition function:

$$T = T(s, a, s') = P(S_{t+20} = s' | A_t = a, S_t = s)$$

Cost function:

C = C(s, a)

Conclusions

In conclusion, we have found this project pretty enjoyable, refreshing, and useful. We consider that it has allowed us to dive deeper into the more practical aspects of the subject and to expand our knowledge of the topics treated and programming of algorithms as a whole.

Although we had some doubts about the resolution at some points, we managed to solve it without any problems by dividing the problem into several sub-problems which were easier to understand and to deal with.

Budget

The estimated budget has been calculated by taking into account the time spent on the completion of the project and the estimated salary.

The approximate time employed on solving this project, in which we include all the time spent planning how to approach it, designing the model, developing the code, and completing all media associated such as the report and the video, is of around 20 hours in total. The base starting salary for the IA field in Spain is estimated between 30.000€ and 60.000€ each year. As we are ‘starting developers’ we will take the lowest one and state the base salary per hour at 62,5€/h.

Given the data above and that the project was developed by two people, we can calculate the project's budget to be 2500€ net.