AI Final Project - Report

Ginevra Cepparulo, Nicholas Froehlinger

Executive summary

The logistics of traffic and mass transit systems are highly complex. Since traffic patterns may be influenced by a wide array of factors, it can be prohibitively difficult to plan efficient policies for traffic-control decisions if relying solely on manual analysis.

To address the matter, for the intersection in question, we have employed artificial intelligence in order to generate, based upon historical data on traffic patterns in the intersection, a policy by which the intersection's lights may be controlled in order to greatly improve traffic flow.

The system relies upon the sensory capacity to identify whether incoming traffic in each of three ingress lanes (North, East, and West) is at a *High* or *Low* volume. Given this situational (state) information, the computed policy identifies which of those three lanes ought to receive a green light in order to best alleviate traffic. Specifically, the policy is optimized for the following goal: achieve low traffic volumes in all three ingress lanes in as little time as possible.

The policy was derived by applying a probabilistic understanding of traffic based on historical data. The cost function was determined in order to most directly align with the goal of reaching Low;Low;Low traffic as efficiently (in terms of time) as possible. The system designed, however, is more than capable of supporting changes should economic considerations not previously mentioned be brought into consideration. For example, if traffic delays in a certain direction carry a heavier financial burden for the city, the model can readily optimize a new policy which instead minimizes the financial burden of traffic in a way which accounts for such factors.

This extension, and other possibilities, are expanded upon further in the Extended Results section of the report.

Objectives

The objective of our project consists in finding a solution for the traffic control problem. Recalling the problem presented to us; a traffic intersection can be approached by the North, East and West directions, where each direction is controlled by a stop light. In every 20-second cycle, incoming traffic levels are measured in each direction and one stop light is turned green while the other two are automatically turned red. The goal is to achieve low traffic in all directions while incurring minimum cost.

The problem can be represented by a Markov Decision Process (MDP), wherein traffic conditions are treated as state information and traffic-control decisions (which lane receives a green light) are possible actions. Thereby, a solution to the problem involves finding an optimal policy for this MDP that can determine, for each 20-second cycle, which stop light in the intersection should be turned green so as to achieve low traffic in each direction of the intersection with minal cost. We treat the cost of each traffic-control action from each state as being equal. Essentially, the cost of any decision is the 20 seconds of time until traffic is measured again.

This cost model was chosen because we believe it the most reasonable representation of what happens in real life; traffic lights are always shining, whether red or green, so any combination of red and green lights should incur an equal material cost in terms of resources (electricity, physical wear on light bulbs, etc.). With this cost model, the goal of the MDP, for which the resultant policy is optimized, is to achieve low traffic in all lanes as quickly as possible. With a different cost model, the policy might instead be optimized for some metric other than this one.

Formal description of the AI model, including all its elements and the corresponding graphical representation.

The AI model used is a Markov Decision Process defined by the following:

- **States:** States represent the levels of traffic in each direction: North, East, and West. States, then, are composed of the traffic level of all three ingress lanes, and are thus denoted by strings of the format tN;tW;tE wherein *t*N represents the level of traffic entering from the North, *t*W for traffic from the West, and *t*E from the East. Each substring element (tN, tW, and tE), takes a value from the domain {High, Low}. There are 8 total states:
    - High;High;High
    - High;High;Low
    - High;Low;High
    - High;Low;Low
    - Low;High;High
    - Low;High;Low
    - Low;Low;High
    - Low;Low;Low
- **Actions:** Actions represent the direction of the stoplight that is turned green. They are denoted by a letter from the domain {“E”, ”W”, “N”} . Where “E” represents the East direction, “W” the West direction, and “N” the North direction.
- **Transition function:** The transition function T(s,a,s') → P returns the probability that taking action *a* while in state *s* will result in a transition to state *s*’. The transition function was obtained by simple experimental probability from historical data gathered at the intersection.
- **Cost function:** The cost function C(s,a) → R returns the cost of making a transition from state *s* following action *a*. No cost function or metric was provided by the problem statement. For this MDP, we employ a cost function wherein each action simply costs the 20 seconds of time that will pass until traffic state is measured again. The rationale for this representation is described in our Objectives.

Methodology

In order to utilize a Markov Decision Process and compute an optimal policy, one must establish the states, actions, transition function, and cost function. After determining during planning stages the states, actions, and cost function to be used, we needed to first generate the transition function, which for us, meant parsing the provided CSV-formatted data of historical traffic patterns.

Provided data identifies the pre-action traffic conditions, action taken, and resultant conditions for various recorded 20-second intervals. Using regular expressions, this CSV-formatted data could be separated into three capture groups: traffic state, action taken, and output state. The traffic conditions of all three lanes were taken as a compound traffic state, as per the formalization of the MDP.

Once we were able to read and parse the format of the data, the next step was to verify that we possessed historical data for every action taken in every state other than Low;Low;Low. If we had no performance data for the results of each action at each initial state, we would be unable to derive a transition table, as no experimental data would exist for certain state-action combinations. Fortunately, we ran test code to count the total initial-state:action pairings in the data, and 7*3=21 were identified.

To compute the experimental probability that any action in any state would have to reach any other state, our code tabulates, for each initial-state:action pair, the incidence of each output state. Next, each count is divided by the total incidence of data from that initial state, yielding percentage probabilities. Thus, we produced a dictionary mapping Strings (input-state string formatting concatenated with action strings) to an inner dictionary which mapped Strings (output-state string formatting) to Floating Point Numbers (probabilities).

Using this data structure, which is saved as JSON to a file for reading and caching, our transition function for the model needs only perform simple map lookups into this transition table.

The implementation of the policy- and value-iteration algorithm was absolutely standard. The code features calls out to separate functions which implement the cost and transition functions for the sake of modularity, legibility, and modifiability. Since value-iteration inherently generates the expected costs of each state, while recording the policy decisions yields the optimal policy, the outputs of this function are two dictionaries, the expected costs table and optimal policy. Both of these are also saved as JSON files for caching and reading.

In order to test performance, the function *tracerun_MDP* was written, which for a configurable number of trials will simulate use of the optimal policy by the traffic-control MDP, all the while recording the states reached and cost incurred. The costs of each performance from each starting position are summed and divided by the trial count to yield average costs, which are then printed to the display to be manually compared with the previously computed expected costs. When run with 30,000 trials, performance matched expectations quite well, with no performance average varying by more than small fractions of a second in terms of our time cost function.

Resources used (Python and any other)

In development and analysis of this traffic-control MDP, the only technical resource utilized was Python and its standard import libraries. The traffic-control system MDP, historical-data parsing, policy computation, and value estimation were all implemented purely with Python. All analysis utilized knowledge gained from our individual participation in UC3M's Artificial Intelligence course and from selected works enumerated in IEEE format in Works Cited. In order to version-control, merge, manage, and house our codebase, we utilized GitHub and the Git protocol.

Development (activities: design, implementation, testing ...)

Our development stages included: design, implementation, testing and improvement.

During the design stage, we modeled the problem and highlighted the tasks that needed to be done in order to accomplish the objective. The outcomes of the design stage included the Formal Description of the MDP as well as the following list of tasks:

- Obtain the transition function from the data set and store it in a data structure

- Implement a value- and policy-iteration algorithm for our problem to obtain the optimal policy and the expected costs based upon our transition and cost functions
- Develop a simulation of the MDP running from all possible initial states following the optimal policy in order to trace their operation.
- Using the simulation, consistency-check our model's performance (average execution cost across thousands of trials) against previously-computed expected costs

During the implementation stage, we developed our program in Python. We divided each task into smaller ones and created separate functions which, when coordinated, provide a solution. When the program is run, the function main() is called, which acts according to the command line arguments provided by the user. Commands can be provided of two main forms:

- python input_parser.py Create {no,splitdays} *input_data_file.csv*
    - e.g.: python input_parser.py Create no Data.csv
- python input_parser.py Load {no, splitdays}
    - e.g.: python input_parser.py Load no

When the program is running in “Load” mode, the file names provided as input are already populated with data because the program was previously run and had created those files. Therefore the main() function will open those files, read-in their pre-computed data structures, and call the function tracerun_MDP(), which is in charge of running a simulation of the MDP using the transition table, expected_costs and optimum policy provided as input.

Otherwise, running in “Create” mode, the program will need to make some more computations in order to populate the files with information. Therefore, main() will call the function make_transition_table() to compute the transition table from the provided data CSV file. Then, it will perform the policy iteration algorithm by repeatedly calling the function value_iteration() from a loop in main() until a policy is finalized. In the process of computing the optimal policy, the algorithm also computed the expected costs from each initial state. This means that we do not need to backtrack or perform any further work in order to compute the expected costs. In other words, the function value_iteration() computes the inner loop of the algorithm; finding a policy and cost from an initial state. The main function computes the outer loops; running value_iteration() for each state until all the expected costs stabilize.

In value_iteration() the function actions() and cost_function() are called. The former returns the possible actions from a given state. Of course it was assumed that all states allow all possible actions to be performed; “N” or “W” or “E”, except in the goal state, where it was requested that the MDP looped over the possible actions indefinitely. The latter returns the cost of performing an action from a given state given as parameters. Of course, in our program this was a “dumb” function because it always returns a number 20; which represents the time between the action and the next state being received as input.

During the testing, we mainly checked if the costs from our simulation of the MDP were similar to the expected costs achieved through value iteration. We ran the simulation several times, with an increasing number of iterations, the results converged more and more to the value iteration ones. We were satisfied with 30,000 iterations of the MDP simulation.

Results (Q1 – Q5 above)

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?
    a. Because the problem statement was designed around finding the optimal policy to reach Low-Low-Low traffic as an effective goal state, it makes sense that we wouldn't require data for any transitions out of that state. Fundamentally, our Markov Decision Process (MDP) is built around reaching this goal state, and it's designed to optimally reach that state, using a policy derived from historical traffic data.
    b. If the problem were an essentially different one, such as "minimize the incidence of High traffic over a long period of time", then historical traffic data would ideally include transitions out of every state, including Low-Low-Low. In such a case, the cost function might focus instead on penalizing actions taken out of High-traffic states, and the optimal policy, rather than trying to reach Low-Low-Low above all else, might instead optimize for any other metric.
    c. Because our MDP doesn't have any calculated policy for what to do out of Low-Low-Low, it must follow some default action sequence while in this state. This is why our traffic-control program, when in Low-Low-Low, will rotate the green light between North, East, and West for as long as the state remains unchanged. If we had data about transitions out of Low-Low-Low, however, we would be able to compute a policy and determine the optimal action that might preserve that state.
2. The statement does not say anything about the cost of the actions. What reasonable assumption could we make?
    a. When it comes to traffic lights, the physical lights must always be on, whether that is the bulb behind a red or a green lens. The controller must always be computing the next decision. So, the same lights are on, and the same circuits are in use, regardless of the decision being made. We also have no hysteresis or long-term data to keep in mind, since we must abide by the Markov Principal for this MDP, so we can't reasonably try to reduce manual labor costs by making the red and

green light bulbs wear out around the same time or anything akin to that.

b. When the goal is simply to reach Low-Low-Low traffic as quickly as possible, we feel the best cost function is one in which all actions from all states cost the same: 1 "cost unit". In this way, the real cost incurred by any action is the 20 seconds that will pass until the next action is taken, repeating until the end/goal state is reached. Because of this, optimal policies reach Low-Low-Low traffic in the minimum number of steps.
c. With any other cost function, given the same problem statement and goal, you would interfere with computation of the fastest probabilistic path to Low-Low-Low; this could absolutely be legitimate, such as if having the North-light be green is more costly to the city, but without any information indicating this, it would be imprudent to assume such a situational factor that would sway policy.
d. If the goal of the MDP were different, for instance if the policy were optimized to reduce the total seconds spent in High traffic across all lanes, then it might make more sense to attribute greater cost to moves which delay traffic, such as any action where High traffic is kept at a red light.

3. What are the expected values of the states? Provide the values to six decimal places. The precision must be greater than one thousandth.
    a. We take this question to mean "taking into account the assumptions you made for how to generate the optimal policy with your chosen cost function, what is the expected cost to read Low-Low-Low traffic beginning from each state?"
    b. For this list of expected costs, costs are given in terms of seconds spent making decisions until Low-Low-Low is reached. This is equal to 20 seconds multiplied by the number of actions taken between the initial and goal state. Traffic states are identified by the traffic state (Low or High) for each input lane, in North;East;West order.
        i. High;High;High: 794.2729090013163 seconds
        ii. High;High;Low: 739.0091378449082 seconds
        iii. High;Low;High: 739.0091378449082 seconds
        iv. High;Low;Low: 587.4027533581908 seconds
        v. Low;High;High: 745.3604549119137 seconds

vi. Low;High;Low: 615.2237214721317 seconds
vii. Low;Low;High: 593.1620220754028 seconds
viii. Low;Low;Low: 0 seconds

4. What is the optimal policy?
    a. How this was computed: first, compute the expected cost to reach the goal from each state using a Value-Iteration Algorithm (VIA). The expected cost of each action at each iteration is calculated by weighting the expected costs of each output state according to transition probabilities, then adding the action cost. The expected cost is taken as the minimum-expected-cost action. The VIA can be converted into a Policy-Iteration Algorithm (PIA) which action was used to determine the expected cost during each iteration.
    b. From each state (enumerated according to traffic conditions in North;East;West order), the optimal green light direction is as follows:
        i. High;High;High: E
        ii. High;High;Low: E
        iii. High;Low;High: W
        iv. High;Low;Low: N
        v. Low;High;High: E
        vi. Low;High;Low: E
        vii. Low;Low;High: W
        viii. Low;Low;Low: no policy applies, simply rotate the green light between North, East, and West
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?
    a. Currently, traffic states consist of a binary traffic level, Low or High, for each of three directions. This means that there are 2^3=8 possible traffic states. For actions, 3 exist, one for each lane of traffic. This means that there are transition probabilities from 8 states, across three actions, to 8 possible output states. The transition table, then, is stored in $8^2 * 3 = 192$ different probability values (stores as floats).
    b. Introducing a 4th traffic lane, 4th action, and levels 3-5 of traffic would result in: 5^4=625 states, with 4 actions. The transition table, then, would express the probability from each of 625*4 state:action pairs to each of 625 possible output states. This would be $625^2 * 4 = 1562500$

different floating point numbers, each indicating the probability that some action *a* from some state *s* would transition to some state *s'*.

c. Practically speaking, such a system would be able to more finely and accurately account for subtler traffic trends. Especially important is that, while the Markov Principle prevents hysteresis or accounting for runtime history, having non-binary traffic levels for each lane would allow for a policy that addresses traffic growth before it reaches the highest known state.
d. Runtime, once the optimal policy was determined, would not significantly change, as the time complexity of map lookup is constant-time. The computational time to derive that optimal policy, however, would greatly expand, because the value-iteration algorithm has a time complexity, for each iteration, of $O(S^2*A)$ for a state space of size S and A actions at each state. The number of iterations required is exceedingly computationally complex to describe but notably might increase with increases to S and/or A (https://arxiv.org/pdf/1807.04920.pdf).
    i. In the initial scenario, as described, S=8 and A=3; each VIA iteration involves $8^2*3=192$ computations; in the new case, $S^2*A = 1562500$, as discussed above, marking a non-negligible computational increase even before the risk of iteration-expansion is broached

Extended Results (further lines of questioning)

1. Can policies be generated to address more fine-grained scenarios, such as considering morning v.s. evening traffic patterns, or different days of the week?
    a. Yes! The process of generating transition tables based upon historical data is implemented as a function which requires a correctly-formatted CSV file as input. In our implementation we have allowed the user to select if they would like to split the data set into days by using the “splitdays” keyword in command line arguments. If this keyword is used in the command line then the data set will be split in 3 days.

    b. Our results about dividing the data set into three days have shown us that the expected costs for day1 were generally much lower than expected costs of the next two days, meaning perhaps that day was generally a more traffic intense day of the week.
2. Can the model be optimized for a different goal, for instance, minimizing total traffic, rather than achieving Low;Low;Low as quickly as possible?
    a. Yes! The Cost Function is the representation of the metric(s) for which the model optimizes its policies.
    b. At the top of input_parser.py, a configuration value *cost_mode* is set to "Equal". If this is changed to "MinTraffic", an alternative cost function is utilized.
    c. This alternative cost function assigns cost according to the total incidence of High traffic in the source state. Specifically, cost increases exponentially with the number of High-traffic lanes. With this model, actions might be taken that are less optimal at reaching Low;Low;Low traffic efficiently, but which are more effective at reducing the total incidence of High traffic while progressing towards Low;Low;Low.
        i. Interestingly, this alternative cost model, with this data set, produces an identical optimal policy!
        ii. For an example of another possible cost model which results in a different policy, cost_mode "ImportantE" exclusively penalizes Low;High;Low traffic massively but treats all other conditions, except Low;Low;Low as being equal in cost. This incentivizes a policy optimized around reaching Low;Low;Low while desperately avoiding Low;High;Low traffic in particular. As a result, the generated policy is as follows:
            1. High;High;High: E
            2. High;High;Low: E
            3. High;Low;High: W
            4. High;Low;Low: N
            5. Low;High;High: E
            6. Low;High;Low: E
            7. **Low;Low;High: E**
            8. Low;Low;Low: no policy applies, simply rotate the green light between North, East, and West

Conclusions (technical comments related to the Project development and personal comments: difficulties, challenges, benefits, etc.)

We believe that the project was fairly interesting and challenging. We can agree that the most interesting part entailed representing the proposed problem with an AI model. There were a lot of important decisions, such as the actualization of problem goals into the cost model, that held a significant influence on the results. This, for us, provided an interesting takeaway: optimization is not absolute; rather, optimality exists relative to a specific mathematical model of some given variables (metrics). Using the same essential algorithms and approaches, many different problems can be solved, or rather, different goals achieved.

It was also tricky to glue all the pieces of the program together. We didn't encounter many challenges in working together, but as a group programming project, it was at times difficult to organize the programming between both members of the team. GitHub and the ability of Git to merge changes was indelible.

Budget (economic estimate of how much the study and implementation would have cost if someone had contracted it out to you.)

Hours spent on the project, including design, implementation, testing, and report: 20h
Price per hour: 50 euros per hour
Total Cost of hire for the job of 2: 20*50*2 = 2000

Works Cited

[1] Bala et al., "On the Complexity of Value Iteration," arXiv:1807.04920v3 [cs.FL], 27 Apr. 2019. https://arxiv.org/pdf/1807.04920.pdf. [Accessed 10 May 2022].