Machine Learning and Data Mining Task 2: Roadster - The Learning Traffic Light Simulator

Benjamin J. Wright, Christopher J. Di Bella, Daniel Playfair Cal

2013, June 9, 23:59:59

1 Introduction

Todays world revolves around the ease of use. As we have become dependent on technology, there is an ever-increasing need to improve systems that support the safety of human lives, so that they become more efficient.

Traffic lights are one such system. Current Australian traffic light systems are beginning to struggle under the sheer volume of traffic that occupies our roads. The objective of this task is to simulate a busy intersection with a learner that adjusts the lights as the roads become more dense. The advantage of simulating traffic using this learner is that it utilises reinforcement learning and thus receives a grading for its decisions.

1.1 Well-posed learning problem

Before detailing the implementation and experimentation, it is important to have a formal definition for our learning problem:

Task *T*:

Performance measure *P*: The total time spent by any car waiting at a light.

Experience *E*:

2 Apparatus

This task was implemented using a perfect subset of C++11. The program also makes use of the *Irrlicht Graphics Library* and the *Boost Graph Library*. Both third party code libraries are properly referenced at the end of this document.

The data was modelled using a combination of the coded simulator and XML files.

3 Implementation

The task makes use of the strategy design pattern and is split into three main sections: learning, simulating, and, rendering.

3.1 Data model

The implementation for the model can be found in `State.hpp` and `State.cpp`.

The best way to mentally picture the model (i.e. without running the program) is to imagine a cartesian plane with horisontal traffic moving along the x-axis from $(-x, 0)$ to $(x, 0)$, and, vertical traffic moving along the y-axis from $(0, y)$ to $(0, -y)$, where $x, y \in \mathbb{R}$. The intersection is at the origin.

Roadster traffic lights are dictated by a binary control system that will either allow horisontal or vertical traffic to flow. A bidirectional graph is used to represent the roads: verticies are used to represent the towns and cities that act as sources (where cars come from), sinks (where cars go to) and intersections (to control flow between corresponding edges); edges consist of double-ended queues (`deque`) that store the `Car`s.

A `Car` is little more than an absolute position relative to the road the car is travelling on and the speed at which it travels.

3.2 Learning policy

Traffic light reinforcement learning is achieved using the Q-learning algorithm. Due to the complexity of explicitly storing the optimal action for each state, it is calculated whenever it is needed by finding the action yielding the maximum reward. This still operates at close to constant-time, as there are only two possible traffic light states to consider per intersection.

Since this problem is a continuous task (that is, it has no end state), a discount factor is applied to expected future rewards when calculating the action-value function and operates at a rate of 0.9. An exploration rate is also used to ensure that all states are explored, operating at a ten-percent probability to choose a random action instead of the action with the highest possible reward. Finally, this problem is non-deterministic, and so a ten-percent learning rate of is used; only a portion of the action-value function is updated after each action is made.

The implementation involves an array of rewards for each state/action pair with enough memory to store all possible states within defined maximum values for the number of lanes approaching an intersection, leaving an intersection and the number of light values. States that are ignored by the learner would not affect the learning process as no action leads to their state. Due to the simplicity of the state and the resulting small number of states, there are a small number of actions.

Internally, they are represented as an integer between zero and the number of states. The learner's state is a subset of the simulator's state (discussed below) and is a composition of the light setting, the number of ticks since the last change (no more than three ticks), and, the distance to the closest car in each lane. This distance is represented by the numbers zero through to nine. Distances up to and including eight represent the actual distance between the car; nine is used to represent distances beyond eight.

While cars are waiting at red lights, the reward function applies the negative sum of all cars queued as its reward; whenever no cars are waiting, the reward function returns a zero reward.

3.3 Simulating policy

3.3.1 Naive policy

The purpose of this part of the system is to provide the learner with data. This task employs reinforcement learning; the system shall perform best when utilising online learning in place of active learning. The simulator provides the all the necessary data for the learner to create its own hypothesis function.

The traffic light state is first updated according to the learner's hypothesis. Any traffic that cannot safely stop will continue to pass through the intersection (usually about three 'ticks' worth of traffic). Following this, traffic is then allowed to pass. Traffic that is heading in the direction of the active traffic light state (horisontal traffic = 0; vertical traffic = 1) passes from the back of the source `deque` to the front of the corresponding sink `deque`. Only the active source's back-most car is actually transferred at any time step. All sinks' back-most car leave the system for good by being popped from the front of the `deque`.

3.3.2 Cellular policy

Although not implemented to completion, the simulation system also supported a second policy that would model a slightly modified version of the Nagel-Schreckenberg model. The key difference was that cars operating at a speed of 1 or less would not be allowed to stop unless absolutely necessary (e.g. a red light or the car in front of them stopped), as opposed to being allowed to randomly stop. `Car`s operating at higher speeds are still able to do this.

This model would have provided a more realistic model of our traffic system, as cars do not behave naively, but due to its complexity, was abandoned in favour of ensuring that the Naive policy is absolutely correct.

3.4 Rendering policy

As a traffic light simulator is best observed using a graphical user interface, *Roadster* employs a three dimensional world to show how the system works. Additionally, the interface provides a WYSIWYG to provide users with a way of creating their own intersections.

`GraphicsPolicy3D` is a front-end to the *Boost* `AdjacencyList` and is able to read, and, write intersection files stored as XML files. To render, the rendering system converts the `AdjacencyList` to a `SceneGraph`. The system also keeps track of all dynamic nodes (which, unlike vertices are cars and intersection lights), and, cleans the cars up once they have arrived at a sink via an internal resource manager.

4 Experiment

4.1 Method

4.2 Results

4.3 Discussion

5 Conclusion

6 References