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*:

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) and sinks (where cars go to); edges consist of double-ended queues (`deque`) that store the `Car`s.

`Car` is little more than an indicator of whether or not a car exists or if there should be an empty space.

3.2 Learning policy

3.3 Simulating policy

The purpose of this part of the system is to show how the learner works. 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 front of the source `deque` to the back of the sink `deque`. Only the active source's front-most car (or `noCar` if it is an empty space) is actually transferred at any time step. All sinks' front-most car leave the system for good by being popped from the front of the `deque`.

The simulator has a forty-five percent chance to introduce a new car into each road after updating the previous cars' positions. Due to the fact that each road is given a chance to introduce a new car, the probability to introduce a new car must be low enough that it does not introduce too many cars too quickly (and thus consume too much memory), but high enough that cars will be introduced often enough for the learner to update the system.