Creation of a Traffic Signal Simulation and Proposal of Throughput Optimization Strategies

Scott Hyndman
Computer Systems Lab

June 14, 2005

Abstract

Traffic problems are a major issue in many cities throughout the United States. Optimizing traffic signals at intersections would help this situation. One way to create an optimization algorithm for a traffic signal is to create a computer model of an intersection. Traffic signal light changing algorithms can then be created and tested on the model in order to create an algorithm that will maximize the traffic flow through the intersection. In this project, a simulation was created on which could be tested signal optimization algorithms. Also included is a proposal of an optimization strategy that could be used to maximize the signal's throughput.

Introduction

This paper discusses the creation of a traffic signal simulation and proposes a traffic signal optimization strategy that could be used with the simulation. The purpose of this project was the accomplishment of the above items. Traffic signals are an integral piece of our road network. They keep traffic moving safely. However, they do not always operate efficiently. Optimizing the throughput of traffic signals would help the gridlock that our roads tend to fall into. This paper includes a discussion of the traffic simulation - its code, logic, etc. - and a possible signal optimization strategy.

Background

1.1 Traffic Signal Control Strategies

There are three main traffic signal control strategies: pretimed control, actuated control, and adaptive control.

1.1.2 Pretimed Control

Pretimed control is the most basic of the three strategies. In the pretimed control strategy, the lights changed based on fixed time values. The values are chosen based on data concerning previous traffic flow through the intersection. This control strategy operates the same no matter what the traffic volume is.

1.1.3 Actuated Control

The actuated control strategy utilizes sensors to tell where cars are at the intersection. It then uses what it learns from the sensors to figure out how long it should wait before changing the light colors. For example, if the signal picks up a car coming just before the green light is scheuled to change, the length of the green light can be extended for the car to go through.

1.1.4 Adaptive Control

The adaptive control strategy is similar to the actuated control strategy. It differs in that it can change more parameters than just the light interval length. Adaptive control estimates what the intersection will be like based on data from a long way up the road. For example, if the signal notices that there is a lot of traffic building up down the road during rush hour, it might lengthen the green light intervals on the main road and shorten them on the smaller road.

1.2 Reinforcement Learning

In reinforcement machine learning is when an agent receives information from the its environment and decides what it must do to reach a goal state. The decision is based on the agent's past experiences - what it has already learned. The agent reviews its end state and figures out how desirable that state is. If the result is undesirable, a negative value is assigned to the course of actions the agent took to get to the end state. If the result is desirable a positive value is assigned.

Development

2.1 MASON

I used the MASON modeling package to create this traffic simulation. MASON is a Java-based modeling package that is distributed by George Mason University. This simulation is based on the MAV simulation included with the MASON download. The following is a brief discussion of MASON classes and interfaces that are important to this project.

2.1.1 Steppable

The basic part of any simulation is that things change over time. One way that a MASON user can make things happen in their simulation is by having a class implement the Steppable interface. Classes that implement Steppable must have the Step function. The user puts actions into the Step function and they are carried out every time the simulation's time iterates. Thus, Steppable creates a way for the simulation to carry out repetetive actions.

2.1.2 Schedule

In MASON, everything runs from the Schedule class. The Schedule keeps track of time and also what objects are running at any given time. It also calls all the Step functions for those that are running and set up with Steppable. One drawback of the Schedule function is that it does not provide a way to make actions happen at specific times. It just repeats the Step functions over and over.

2.1.3 Stoppable

The Stoppable interface that takes objects off the Schedule. Thus, the object's Step function is no longer called. This is done so that the number of objects running does not continually build up and slow down the computer. For example, in the program, when a vehicle has left the intersection area, it is taken off the Schedule. Classes the implement the Stoppable interface contain a Stop

function where the user inputs what is supposed to happen when the class object is taken off of the Schedule.

2.1.4 Continuous2D

A Continuous2D is, as the name suggests, a continuous 2D environment. In truth, it is not continuous, but made up of a grid of densely packed points that are so close together that they essentially make the environment continuous. Continous2D's make it easier to keep track of the objects in a simulation. Objects are placed into a Continuous2D where they can move around and interact with each other. The Continuous2D breaks the space of the simulation into "buckets." If you want to find an object in a certain area of the simulation, you can check in the bucket there. For example, if you want to see if a car has another car near its front, you can look in the bucket that that car is in and check to see where the other cars in that bucket are.

2.2 Simulation

The following is a brief discussion of the what the classes that comprise this program do and how those classes interact to form the traffic simulation.

2.2.1 Car

The Car is the basic piece of this simulation. Each member of the Car class represents a vehicle on the road. Car implements both Stoppable and Steppable. The class contains an agent that controls the Car as it runs. Thus, each Car is autonomous as it makes its way around its environment. All of the Cars in the simulation are members of a Continuous2D. They look in the Continuous2D to find Cars that are near them. If there is a Car too close to them, they will stop to avoid an accident. Cars use the same method to check whether they are coming up on a red light at an intersection.

2.2.2 Region

The Region class is what forms the background of the visual output. Regions are shapes that are placed on the output. The roads, medians, and other background objects are created from many Regions that are combined to make larger, more complex pictures. When the program starts, a text file, 4-way.txt, is read in with a list of all the Regions that will go into the simulation. This is to make it so the program could be more easily expanded to work with different types of intersections. At this time, however, other pieces of the program are not coded in such a way that such flexibility is possible.

2.2.3 Signal

The Signal class is essentially a simplified version of the Region class. The main difference between the Signal and Region classes are that Signals are redrawn every time iteration while Regions are only redrawn if they change location or size.

2.2.4 CarRun

The CarRun class is what creates the simulation. It creates four Continuous2D's, one for each of the object types used in CarRun - Car, Region, Signal, and eventController. CarRun also creates a Schedule and starts each of the objects in the simulation running on it.

2.2.5 CarUI

CarUI, Car User Interface, performs the function of making the simulation visible and letting the user control the running of the simulation. It takes everything CarRun is doing and puts it on the screen.

2.2.6 eventControl

The eventControl class is used to control events in the simulation that do not take place every time iteration. eventControl calls functions defined in other classes to control the objects in those classes. For example, eventControl creates new Cars and tells the Signals when to change color.

Analysis and Conclusion

References

3.1 Traffic Signal Strategies

"Modeling of Traffic Signal Control and Transit Signal Priority Strategies in a Microscopic Simulation Laboratory" by Angus P. Davol

3.2 Driver Behavior

"Volume II: Driver Attitudes & Behavior" by the National Highway Traffic Safety Administration

3.3 Machine Learning

None at this time.

Appendix

5.1 Code

5.1.1 Car

```
   package sim.app.traffic;

   import sim.portrayal.*;
   import sim.engine.*;
   import sim.util.*;
   import sim.field.continuous.*;

// we extend OvalPortrayal2D to steal its hitObjects() code -- but
// we override the draw(...) code to draw our own oval with a little line...

   public /*strictfp*/ class Car implements Steppable, Oriented2D, Stoppable
   {
      public final static double[] theta = new double[/* 8 */]
      {
         0*(/*Strict*/Math.PI/180),
         45*(/*Strict*/Math.PI/180),
         90*(/*Strict*/Math.PI/180),
         135*(/*Strict*/Math.PI/180),
         180*(/*Strict*/Math.PI/180),
         225*(/*Strict*/Math.PI/180),
         270*(/*Strict*/Math.PI/180),
         315*(/*Strict*/Math.PI/180)
      };

      public final static double[] xd = new double[/* 8 */]
      {
        /*Strict*/Math.cos(theta[0]),
         /*Strict*/Math.cos(theta[1]),
```

```
         /*Strict*/Math.cos(theta[2]),
         /*Strict*/Math.cos(theta[3]),
         /*Strict*/Math.cos(theta[4]),
         /*Strict*/Math.cos(theta[5]),
         /*Strict*/Math.cos(theta[6]),
         /*Strict*/Math.cos(theta[7]),
      };

      public final static double[] yd = new double[/* 8 */]
      {
        /*Strict*/Math.sin(theta[0]),
         /*Strict*/Math.sin(theta[1]),
         /*Strict*/Math.sin(theta[2]),
         /*Strict*/Math.sin(theta[3]),
         /*Strict*/Math.sin(theta[4]),
         /*Strict*/Math.sin(theta[5]),
         /*Strict*/Math.sin(theta[6]),
         /*Strict*/Math.sin(theta[7]),
      };

      public int orientation = 0;
      public int direction;
      public boolean active = true;
      public double x; //car's x position
      public double y; //car's y position
      public double speed;
      public double adjSpeed = 0.0;
      public double speedLimit;
      public double accel = 80.5;
      public double decel = 81.0;
      private int wait = 0;
      public int laneType = 0; //1 = left, 2 = center, 3 = right
     // public boolean visible = false; //tells the car whether it is on the screen

      public int minCarDistance;
      public int maxStopDistance;
      public double orientation2D() {
         return theta[orientation]; }

      public Car(int orientation, double x, double y, double limit, int type)
      {
         this.orientation = orientation; this.x = x; this.y = y;
this.minCarDistance = 40;
this.maxStopDistance = 30;
```

```
 this.speed = 35.0;
 this.speedLimit = limit;
 this.laneType = type;
      }
      public double dx = 0.0; //the rate of change in x position
      public double dy = 0.0; //the rate of change in y position
  public void stop()
  {
  active = false;
  }

public boolean frontClear(CarRun carrun, Continuous2D objects, int type)
{
Bag nearbyObjs = objects.getObjectsWithinDistance(
new Double2D(x, y), 100, false, false);
if(type == 0){
for(int i = 0; i < nearbyObjs.numObjs; i++)
{
Car obj = (Car)(nearbyObjs.objs[i]);
if(carrun.sensorRangeDistance *
carrun.sensorRangeDistance >
(obj.x - this.x) * (obj.x - this.x) +
(obj.y - this.y) * (obj.y - this.y))
if(check(this.orientation, this.orientation, this.x, this.y, obj.x, obj.y))
return false;
}
}
else if(type == 1){
for(int i = 0; i < nearbyObjs.numObjs; i++)
{
Signal obj = (Signal)(nearbyObjs.objs[i]);
if(carrun.sensorRangeDistance *
carrun.sensorRangeDistance >
(obj.x - this.x) * (obj.x - this.x) +
(obj.y - this.y) * (obj.y - this.y)
   && obj.orientation == this.orientation)
if(check(obj.surface + 7, this.orientation, this.x, this.y, obj.x, obj.y))
return false;
}

}
return true;
}
```

```
public boolean check(int type, int orientation, double x, double y, double ox, double oy)
{
switch(type)
{
case 4: //moving left
if(x - ox < this.minCarDistance &&x - ox > this.maxStopDistance)
return true;
break;
case 0: //moving right
if(ox - x < this.minCarDistance && ox - x > this.maxStopDistance)
return true;
break;
case 2: //moving down
if(oy - y < this.minCarDistance && oy - y > this.maxStopDistance)
return true;
break;
case 6: //moving up
if(y - oy < this.minCarDistance && y - oy > this.maxStopDistance)
return true;
break;
case 9:
if(check(orientation, orientation, x, y, ox, oy))
return true;
case 10:
if(check(orientation, orientation, x, y, ox, oy))
return true;
break;
default:
return false;
}
return false;
}

      public void step(SimState state)
      {
         final CarRun carrun = (CarRun)state;
       //  orientation += carrun.random.nextInt(3) - 1;

 if (orientation > 7) orientation = 0;
         if (orientation < 0) orientation = 7;
 if(frontClear(carrun, carrun.cars, 0))
 {
 if(frontClear(carrun, carrun.lights, 1))
 {
```

```
 if(speed < speedLimit)
 speed += accel;
 }
 else
 speed -= decel;
 }
 else
 speed -= decel;
 if(speed > speedLimit)
 {
 speed = speedLimit;
 carrun.totalWait -= wait;
 wait = 0;
 }
 else if(speed < 0.0)
 {
 speed = 0.0;
 wait ++;
 carrun.totalWait++;
 }

         adjSpeed = speed / speedLimit;
 x += adjSpeed * xd[orientation];
 y += adjSpeed * yd[orientation];

 if (x >= carrun.width + 20){
            if(orientation == 0){
carrun.controller.delCar = this;
this.stop();}

            //x = carrun.width - 1; orientation = 4;
         }
         else if (x < -20){
            if(orientation == 4){
carrun.controller.delCar = this;
this.stop();
}
//x = 0; orientation = 0;
         }
         else if (y >= carrun.height + 20){
            if(orientation == 2){
carrun.controller.delCar = this;
this.stop();
    }
```

```
// orientation =  6; }
// y = carrun.height - 1; orientation = 6;
         }
         else if (y < -20){
            if(orientation == 6){
carrun.controller.delCar = this;
   this.stop();}
         //y = 0; orientation = 2;
         }
         if(active){
            carrun.cars.setObjectLocation(this,new Double2D(x,y));
            act(nearbyCars(carrun), currentSurface(carrun));
         }
      }

      public void act(double[] sensorReading, int currentSurface)
      {
         if (currentSurface == 100) System.out.println("yo");
      }

      double[] proximitySensors = new double[8];  // all squared values

    /* Re-uses the double[], so don't hang onto it */
public double[] nearbyCars(CarRun carrun)
{
for(int i=0;i<8;i++)
proximitySensors[i] = Double.MAX_VALUE;
final double d = carrun.sensorRangeDistance * carrun.sensorRangeDistance;
final Bag nearbyCars = carrun.cars.getObjectsWithinDistance(
new Double2D(x,y),16,false,false);
for(int i=0;i<nearbyCars.numObjs;i++)
{
final Car car = (Car)(nearbyCars.objs[i]);
final double carDistance = (car.x-x)*(car.x-x)+(car.y-y)*(car.y-y);
if (carDistance < d)  // it's within our range
{
final int octant = sensorForPoint(car.x,car.y);  // figure the octant
proximitySensors[octant] = Math.min(
proximitySensors[octant],carDistance);
}
}
return proximitySensors;
}
```

```
/** 0 is the default surface */
  public int currentSurface(CarRun carrun)
  {
     for(int i = 0; i < carrun.region.length;i++)
        if (carrun.region[i].area.contains(
                              x-carrun.region[i].originx,y-carrun.region[i].originy))
           return carrun.region[i].surface;
     return 0;
  }


// in order to rotate 45/2 degrees counterclockwise around origin
  final double sinTheta = /*Strict*/Math.sin(45.0/2*/*Strict*/Math.PI/180);
  final double cosTheta = /*Strict*/Math.cos(45.0/2*/*Strict*/Math.PI/180);

  public int sensorForPoint(double px, double py)
  {
     int o = 0;
    // translate to origin
     px -= x; py -= y;

    // rotate 45/2 degrees counterclockwise about the origin
     final double xx = px * cosTheta + py * (-sinTheta);
     final double yy = px * sinTheta + py * cosTheta;

    // Now we've divided it into octants of 0--45, 45--90, etc.
    // for each sensor area.  The border between octants is
    // arbitrarily, not evenly, assigned to the octants, because
    // it results in fewer if/then statements/

     if (!(xx == 0.0 && yy == 0.0))
     {
        if (xx > 0)         // right side
        {
           if (yy > 0)     // quadrant 1
           {
              if (xx > yy)  o = 0;
              else o = 1;
           }
           else            // quadrant 4
           {
              if (xx > -yy) o = 7;
              else o = 6;
           }
```

```
        }
        else                // left side
        {
           if (yy > 0)     // quadrant 2
           {
              if (-xx > yy)  o = 3;
              else o = 2;
           }
           else            // quadrant 3
           {
              if (-xx > -yy) o = 4;
              else o = 5;
           }
        }  // hope I got that right!
     }

    // now rotate to be relative to MAV's orientation
     o += orientation;
     if (o >= 8) o = o % 8;
     return o;
  }

}
```

5.1.2 Region

```
package sim.app.traffic;

import sim.portrayal.*;
import sim.util.*;
import java.awt.geom.*;
import java.awt.*;
import java.io.*;
import java.awt.font.*;
import sim.engine.*;

public /*strictfp*/ class Region extends SimplePortrayal2D
{
 // we hard-code the available shapes here.  The reason for this is simple: shapes
 // and areas aren't serializable.  ARGH.  So we can't save out a shape/area and
 // load it back in again.  Instead we have to save out a shape "number", and then
 // load that number back in again.  Suboptimal.

   public static final Shape[] shapes = new Shape[] //array of shape types
```

```
    {
       new Rectangle2D.Double(0,0,750,750), //type 0 is a big rectangle
       new Rectangle2D.Double(0,0,200,750), //type 1 is a long, vertical rectangle
       // AffineTransform.getRotateInstance(35*/*Strict*/Math.PI/180).createTransformedSh
       //   new Rectangle2D.Double(0,0,100,100)),
       new Rectangle2D.Double(0,0,750,200), //type 2 is a long, horizontal rectangle
       new Rectangle2D.Double(0,0,10,20), //type 3 is a small, vertical rectangle
       new Rectangle2D.Double(0,0,20,10), //type 4 is a small, horizontal rectangle
       new Rectangle2D.Double(0,0,10,10),//5
new Rectangle2D.Double(0,0,60,220),//6
new Rectangle2D.Double(0,0,220,60),//7
new Rectangle2D.Double(0,0,100,275),//8
new Rectangle2D.Double(0,0,275,100),//9
new Arc2D.Double(0, 0, 175, 150, 0, 90, 2),//10
new Arc2D.Double(0, 0, 150, 150, 90, 90, 2),//11
new Arc2D.Double(0, 0, 150, 150, 180, 90, 2),//12
new Arc2D.Double(0, 0, 150, 175, 270, 90, 2)//13
    //   new Font("Serif", 0, 128).createGlyphVector(new FontRenderContext(
                                       //               new AffineTransform(),false,true)
    };

   // the location of the object's origin.
     public double originx;
     public double originy;
     int shapeNum;

  //an array of possible shape colors
     public static final Color[] surfacecolors = new Color[] {
 Color.green.darker(), //1
 Color.black, //2
 Color.black, //3
 Color.white, //4
 Color.green.brighter(), //5
 Color.yellow.brighter(), //6
 Color.red.brighter(), //7
 Color.yellow//8
 };
     public Shape shape;
     public Area area;
     public int surface;
     public Region (int num, int s,
                    double x,
                    double y) {
        shapeNum = num;
```

```
    shape = shapes[shapeNum]; surface = s;
    area = new Area(shape); originx = x; originy = y; }


// rule 1: don't fool around with graphics' own transforms because they effect its clip
// so we have to create our own transformed shape.  To be more efficient, we only trans
// it if it's moved around.
  Shape oldShape;
  Rectangle2D.Double oldDraw = null;
  public void draw(Object object, Graphics2D graphics, DrawInfo2D info)
  {
     if (oldDraw == null ||
        oldDraw.x != info.draw.x ||
        oldDraw.y != info.draw.y ||
        oldDraw.width != info.draw.width ||
        oldDraw.height != info.draw.height) // new location or scale, must create
     {
        oldDraw = info.draw;
        AffineTransform transform = new AffineTransform();
        transform.translate(oldDraw.x, oldDraw.y);
        transform.scale(oldDraw.width, oldDraw.height);
        oldShape = transform.createTransformedShape(shape);
     }

    // okay, now draw the shape, it's properly transformed
     graphics.setColor(surfacecolors[surface - 1]);
     graphics.fill(oldShape);
  }

/** If drawing area intersects selected area, add to the bag */
  public void hitObjects(DrawInfo2D range, Bag putInHere)
  {
     AffineTransform transform = new AffineTransform();
     transform.translate(range.draw.x, range.draw.y);
     transform.scale(range.draw.width, range.draw.height);
     Shape s = transform.createTransformedShape(shape);

     if (s.intersects(range.clip.x, range.clip.y, range.clip.width, range.clip.height))
        putInHere.add(this);
  }

// because we're using Areas, and for some bizarre reason Area isn't serializable,
// if WE want to be serializable or externalizable we need to handle our own read
// and write methods.
```

```
    private void writeObject(java.io.ObjectOutputStream p)
    throws IOException
    {
       p.writeDouble(originx);
       p.writeDouble(originy);
       p.writeInt(shapeNum);
       p.writeInt(surface);
    }

    private void readObject(java.io.ObjectInputStream p)
    throws IOException, ClassNotFoundException
    {
       originx = p.readDouble();
       originy = p.readDouble();
       shapeNum = p.readInt();
       System.out.println(shapeNum);
       surface = p.readInt();
      // reload shapes and areas, which aren't serializable (ugh)
       shape = (Shape)(shapes[shapeNum]);
       area = new Area(shape);
    }

}
```

5.1.3 Signal

```
package sim.app.traffic;

import sim.portrayal.*;
import sim.util.*;
import java.awt.geom.*;
import java.awt.*;
import java.io.*;
import java.awt.font.*;
import sim.engine.*;

public /*strictfp*/ class Signal extends SimplePortrayal2D
{
 // we hard-code the available shapes here.  The reason for this is simple: shapes
 // and areas aren't serializable.  ARGH.  So we can't save out a shape/area and
```

```
    // load it back in again.  Instead we have to save out a shape "number", and then
    // load that number back in again.  Suboptimal.


    // the location of the object's origin.
      public double x;
      public double y;

   //an array of possible shape colors
      public static final Color[] surfacecolors = new Color[] {
  Color.green.brighter(), //1
  Color.yellow, //2
  Color.red.brighter(), //3
  Color.black
  };
      public Shape shape;
      public Area area;
      public int surface;
      public int orientation;
      public Signal (int s,
                     double ox,
                     double oy,
    int o) {
          shape = new Rectangle2D.Double(0,0,10,10);
 surface = s;
          area = new Area(shape); x = ox; y = oy;
          orientation = o;}


    // rule 1: don't fool around with graphics' own transforms because they effect its clip
    // so we have to create our own transformed shape.  To be more efficient, we only trans
    // it if it' moved around.
      Shape oldShape;
      Rectangle2D.Double oldDraw = null;
      public void draw(Object object, Graphics2D graphics, DrawInfo2D info)
      {
/*          if (oldDraw == null ||
              oldDraw.x != info.draw.x ||
              oldDraw.y != info.draw.y ||
              oldDraw.width != info.draw.width ||
              oldDraw.height != info.draw.height) // new location or scale, must create
          {*/
              oldDraw = info.draw;
              AffineTransform transform = new AffineTransform();
```

```
            transform.translate(oldDraw.x, oldDraw.y);
            transform.scale(oldDraw.width, oldDraw.height);
            oldShape = transform.createTransformedShape(shape);
      //   }

        // okay, now draw the shape, it's properly transformed
         graphics.setColor(surfacecolors[surface - 1]);
         graphics.fill(oldShape);
      }

   /** If drawing area intersects selected area, add to the bag */
      public void hitObjects(DrawInfo2D range, Bag putInHere)
      {
         AffineTransform transform = new AffineTransform();
         transform.translate(range.draw.x, range.draw.y);
         transform.scale(range.draw.width, range.draw.height);
         Shape s = transform.createTransformedShape(shape);

         if (s.intersects(range.clip.x, range.clip.y, range.clip.width, range.clip.height))
            putInHere.add(this);
      }

   // because we're using Areas, and for some bizarre reason Area isn't serializable,
   // if WE want to be serializable or externalizable we need to handle our own read
   // and write methods.

      private void writeObject(java.io.ObjectOutputStream p)
      throws IOException
      {
         p.writeDouble(x);
         p.writeDouble(y);
         p.writeInt(surface);
p.writeInt(orientation);
      }

      private void readObject(java.io.ObjectInputStream p)
      throws IOException, ClassNotFoundException
      {
         x = p.readDouble();
         y = p.readDouble();
         surface = p.readInt();
orientation = p.readInt();
        // reload shapes and areas, which aren't serializable (ugh)
         shape = (Shape)(new Rectangle2D.Double(0,0,10,10));
```

```
        area = new Area(shape);
      }

   }
```

5.1.4 CarRun

```
   package sim.app.traffic;

   import sim.engine.*;
   import sim.util.*;
   import sim.field.continuous.*;
   import ec.util.*;
   import java.io.*;
   import sim.display.*;
   import sim.portrayal.continuous.*;
   import sim.portrayal.simple.*;
   import javax.swing.*;
   import java.awt.Color;

   public /*strictfp*/ class CarRun extends SimState
   {
      public Continuous2D ground;
      public Continuous2D cars;
      public Continuous2D control;
      public Continuous2D lights;
    //  public static boolean graphics = true;
      public double width = 750;
      public double height = 750;
      public double crashDistance = 8;
      public double sensorRangeDistance = 50;
      public double totalWait = 0.0;
      public double speedLimit = 35.0;
      public double mainPercent = 50.0;
      public eventControl controller; //makes are car to keep track of the number of steps

    // shapes on the ground
      public Region[] region;
      public Signal[] signal = new Signal[]
      {
//top left (L - R)
```

```
  new Signal( 3, 245, 215, 2),
  new Signal( 3, 292, 215, 2),
//bottom right (L - R)
  new Signal( 3, 448, 525, 6),
  new Signal( 3, 495, 525, 6),
//top right (T - B)
  new Signal( 1, 525, 245, 4),
  new Signal( 1, 525, 292, 4),
//bottom left (T - B)
  new Signal( 1, 215, 448, 0),
  new Signal( 1, 215, 495, 0),
  //top left turn
  new Signal( 4, 198, 215, 2),
  new Signal( 4, 339, 215, 2),
  //bottom right turn
  new Signal( 4, 401, 525, 6),
  new Signal( 4, 543, 525, 6),
  //bottom left turn
  new Signal( 4, 525, 198, 4),
  new Signal( 4, 525, 339, 4),
  //top left turn
  new Signal( 4, 215, 401, 0),
  new Signal( 4, 215, 543, 0),
      };

      public CarRun(long seed)
      {
         super(new MersenneTwisterFast(seed), new Schedule(1));
      }

      public CarRun(long seed, double percent)
      {
      super(new MersenneTwisterFast(seed), new Schedule(1));
      mainPercent = percent;
      }

      public void start()
      {
         super.start();
 try
 {
 BufferedReader in = new BufferedReader(new FileReader(new File("4-way.txt")));
 int numRegions = Integer.parseInt(in.readLine());
 region = new Region[numRegions];
```

```
for(int i = 0; i < numRegions; i++){
String temp = in.readLine();
int space = temp.indexOf(' ');
int p1 = Integer.parseInt(temp.substring(0, space));
temp = temp.substring(space + 1);
space = temp.indexOf(' ');
int p2 = Integer.parseInt(temp.substring(0, space));
temp = temp.substring(space + 1);
space = temp.indexOf(' ');
int p3 = Integer.parseInt(temp.substring(0, space));
int p4 = Integer.parseInt(temp.substring(space + 1));
region[i] = new Region(p1, p2, p3, p4);
}
}
catch (Exception e){
e.printStackTrace();
}

        ground = new Continuous2D(width > height ? width : height, width, height);
        for(int i = 0 ; i < region.length; i++)
           ground.setObjectLocation(region[i], new Double2D(region[i].originx, region[i].o

lights = new Continuous2D(width > height ? width : height, width, height);
for(int i = 0; i < signal.length; i++)
lights.setObjectLocation(signal[i], new Double2D(signal[i].x, signal[i].y));

//  schedule.setRepeating(signal[i]);


        cars = new Continuous2D(sensorRangeDistance * 2, width, height);
        createCar();
        controller = new eventControl(); //create timekeeper
        schedule.scheduleRepeating(controller); //start it
     }

 public boolean spaceClear(Continuous2D envire, int x, int y, int o)
 {
 Bag nearbyCars = cars.getObjectsWithinDistance(
 new Double2D(x, y), 15, false, false);
 for(int i = 0; i < nearbyCars.numObjs; i++)
 {
Car obj = (Car)(nearbyCars.objs[i]);
if(sensorRangeDistance * sensorRangeDistance > (obj.x - x) * (obj.x - x) + (obj.y - y) * (o
 return false;
```

```
  }
//   if(obj.orientation == o)
//  ;
//   return false;
  }
  return true;
  }

   //adds a car to cars, the map of cars
      public void createCar()
      {
Car car;
double road = 0;
int direction = 0,
lane = 0,
x = 0,
y = 0,
o = 0;

road = 100 * Math.random();
if(mainPercent > road){
direction = (int)(2 * Math.random() - 0.000001);
if(direction == 1)
direction = 2;
}
else{
direction = 2 + (int)(2 * Math.random() - 0.000001);
if(direction == 2)
direction = 1;
}
lane = (int)(2 * Math.random() - 0.000001);
if(direction == 0){
y = -10;
o = 2;
if(lane ==0)
{
x = 298;
}
else
{
x = 248;
}
while(!spaceClear(cars, x, y, o))
{
```

```
y-=30;
// System.out.println("TOP");
}
}
else if(direction == 2){
y = 760;
o = 6;
if(lane == 0)
{
x = 503;
}
else
{
x = 453;
}
while(!spaceClear(cars, x, y, o))
{
y+=30;
// System.out.println("BOTTOM");
}
}
else if(direction == 3){
x = -10;
o = 0;
if(lane == 0)
{
y = 453;
}
else
{
y = 503;
}
while(!spaceClear(cars, x, y, o))
{
x-=30;
// System.out.println("LEFT");
}
}
else{
x = 760;
o = 4;
if(lane == 0)
{
y = 298;
```

```
}
else
{
y = 248;
}
while(!spaceClear(cars, x, y, o))
{
x+=30;
// System.out.println("RIGHT");
}
}
int laneType = 2;
if(lane == 0)
laneType = 1;
car = new Car(o, x, y, speedLimit, laneType);
         cars.setObjectLocation(car, new Double2D(car.x,car.y));
         schedule.scheduleRepeating(car);
      }

      public static void main(String[] args)
      {
         CarRun cars = null;

        // should we load from checkpoint?  I wrote this little chunk of code to
        // check for this to give you the general idea.

         for(int x=0;x<args.length-1;x++)  // "-checkpoint" can't be the last string
            if (args[x].equals("-checkpoint"))
            {
               SimState state = SimState.readFromCheckpoint(new File(args[x+1]));
               if (state == null)   // there was an error -- it got printed out to the scre
                  System.exit(1);
               else if (!(state instanceof CarRun))  // uh oh, wrong simulation stored in t
               {
                  System.out.println("Checkpoint contains some other simulation: " + state)
                  System.exit(1);
               }
               else // we're ready to lock and load!
                  cars = (CarRun)state;
            }

        // ...or should we start fresh?
         if (cars==null)  // no checkpoint file requested
         {
```

```
            cars = new CarRun(System.currentTimeMillis());  // make a new mavs.  Seed the RN
            cars.start();  // prep the bugs!
            System.out.println("Starting cars.  Running for 50000 steps.");
         }

         long time;
      /* if(graphics)
      {
       CarUI model = new CarUI();
       Console c = new Console(model);
       c.setVisible(true);
      }*/
         while((time = cars.schedule.time()) < 50000)
         {
            // step the schedule.  This is where everything happens.
            if (!cars.schedule.step(cars))
               break;   // it won't happen that we end prematurely,
            // but it's worth checking for!

            if (time%1000==0 && time!=0)
               System.out.println("Time Step " + time);

            // checkpoint
            if (time%5000==0 && time!=0)
            {
               cars.createCar();
               String s = "hb." + time + ".checkpoint";
               System.out.println("Checkpointing to file: " + s);
               cars.writeToCheckpoint(new File(s));
            }
         }

         cars.finish();  // we don't use this, but it's good style
      }
   }
```

5.1.5 CarUI

```
package sim.app.traffic;
```

```
import sim.portrayal.continuous.*;
import sim.portrayal.simple.*;
import sim.engine.*;
import sim.display.*;
import javax.swing.*;
import java.awt.Color;

import java.io.*;

public class CarUI extends GUIState
{
   public Display2D display;
   public JFrame displayFrame;
   public CarRun myCars;

   public static void main(String[] args)
   {
      CarUI car = new CarUI(Double.parseDouble(args[0]));  // randomizes by currentTimeM
      Console c = new Console(car);
      c.setVisible(true);
   }

   ContinuousPortrayal2D obstaclePortrayal = new ContinuousPortrayal2D();
   ContinuousPortrayal2D carPortrayal = new ContinuousPortrayal2D();
   ContinuousPortrayal2D lightPortrayal = new ContinuousPortrayal2D();

   public CarUI()
   {
      super( new CarRun(System.currentTimeMillis()));
   }

   public CarUI(double percent)
   {
   super(new CarRun(System.currentTimeMillis(), percent));
   }

   public CarUI(SimState state)
{
      super(state);
   }

   public String getName() {
      return "Traffic Optimization"; }
```

```
    public String getInfo()
    {
       return "<H2>Traffic Optimization</H2>by Scott Hyndman";
    }

    public void start()
    {
       super.start();
       setupPortrayals();
    }

    public void load(SimState state)
    {
       super.load(state);
       setupPortrayals();
    }

    public void setupPortrayals()
    {
       myCars = (CarRun)state;
      // obstacle portrayal needs no setup
       obstaclePortrayal.setField(myCars.ground);
lightPortrayal.setField(myCars.lights);
       carPortrayal.setField(myCars.cars);
       carPortrayal.setPortrayalForAll(new OrientedPortrayal2D(new RectanglePortrayal2D(2
      // reschedule the displayer
       display.reset();
       display.setBackdrop(Color.white);

      // redraw the display
       display.repaint();
    }

    public void init(Controller c)
    {
       super.init(c);

      // make the displayer
       display = new Display2D(750,750,this,1);

       displayFrame = display.createFrame();
       displayFrame.setTitle("Traffic Optimization Display");
       c.registerFrame(displayFrame);   // register the frame so it appears in the "Displ
       displayFrame.setVisible(true);
```

```
        display.attach( obstaclePortrayal, "Regions" );
        display.attach( carPortrayal, "Cars" );
display.attach( lightPortrayal, "Lights" );
     }

     public void quit()
     {
        super.quit();

        if (displayFrame!=null) displayFrame.dispose();
        displayFrame = null;
        display = null;
     }

  }
```

5.1.6 eventControl

```
  package sim.app.traffic;

  import sim.portrayal.*;
  import sim.engine.*;
  import sim.util.*;
  import java.io.*;

  public /*strictfp*/ class eventControl implements Steppable
  {
     public eventControl(){
     }
     public Car delCar = null;
     public int grnLen = 350;
     private final int yelLen = 350;
     int phaseLen;
     int c = 0;
     int carAdd = 35;
     public boolean maxWait = false;
     public void step(SimState state)
     {
final CarRun carrun = (CarRun)state;
        long time = carrun.schedule.time();
phaseLen = grnLen + yelLen;
c++;
```

```
        if(time % carAdd == 0 && time != 0)
        {
         //  System.out.println("Time Step " + time);
           carrun.createCar();
        }
 if(delCar != null){
carrun.cars.remove(delCar);
delCar = null;
 }
 if(c == grnLen)
 for(int i = 0; i < carrun.signal.length - 8; i++)
 if(carrun.signal[i].surface == 1)
 carrun.signal[i].surface = 2;
 if(c == phaseLen)
 {
 for(int i = 0; i < carrun.signal.length - 8; i++)
 {
 if(carrun.signal[i].surface == 2)
 carrun.signal[i].surface = 3;
 else
 carrun.signal[i].surface = 1;
 }
 c = 0;
 }
 System.out.println("Total Wait: " + carrun.totalWait);

        /*if(time % 5000 == 0 && time != 0)
        {
           String s = "hb." + time + ".checkpoint";
           System.out.println("Checkpointing to file: " + s);
           carrun.writeToCheckpoint(new File(s));
        }  */
     }
  }
```

5.1.7 4-way.txt

```
91
0 1 0 0
8 2 175 0
8 2 475 475
```

9 2 575 425
8 2 425 0
9 2 0 475
9 2 475 175
1 2 275 0
2 3 0 275
9 2 0 225
8 2 225 575
3 4 320 20
3 4 320 60
3 4 320 100
3 4 320 140
3 4 320 180
3 4 270 20
3 4 270 60
3 4 270 100
3 4 270 140
3 4 270 180
3 4 220 20
3 4 220 60
3 4 220 100
3 4 220 140
3 4 220 180
3 4 270 580
3 4 270 620
3 4 270 660
3 4 270 700
3 4 470 20
3 4 470 60
3 4 470 100
3 4 470 140
3 4 420 540
3 4 420 580
3 4 420 620
3 4 420 660
3 4 420 700
3 4 470 540
3 4 470 580
3 4 470 620
3 4 470 660
3 4 470 700
3 4 520 540
3 4 520 580
3 4 520 620

3 4 520 660
3 4 520 700
4 4 20 270
4 4 60 270
4 4 100 270
4 4 140 270
4 4 540 320
4 4 580 320
4 4 620 320
4 4 660 320
4 4 700 320
4 4 540 270
4 4 580 270
4 4 620 270
4 4 660 270
4 4 700 270
4 4 540 220
4 4 580 220
4 4 620 220
4 4 660 220
4 4 700 220
4 4 20 420
4 4 60 420
4 4 100 420
4 4 140 420
4 4 180 420
4 4 20 470
4 4 60 470
4 4 100 470
4 4 140 470
4 4 180 470
4 4 20 520
4 4 60 520
4 4 100 520
4 4 140 520
4 4 180 520
4 4 580 470
4 4 620 470
4 4 660 470
4 4 700 470
6 1 370 0
6 1 320 530
7 1 0 320
7 1 530 370

5.2 Other Papers