The Tragedy of the Commons in Traffic Routing and Congestion

Craig Haseler
Computer Systems Lab
TJHSST
2008-2009

June 4, 2009

Abstract

This project uses Java to create a functional traffic simulation, focusing on routing and congestion rather than individual car physics. We can then use the simulation to make several important conclusions about human behavior. The human tendency to always be self serving is considered an advantage in the economic system of today, but is this also true for other systems? This project could demonstrate the effectiveness of a traffic solution in which a central computer makes decisions rather than individual drivers. While that kind of system is not currently feasible, it will not be long before we will have the technology to implement it on highways at least. In most respects, it will be a simple matter of connecting the cruise control system of cars to a central highway computer bank. Of course, there would be the hurdles of justifying this much control to a computer (and of course the risks), but this project should demonstrate that turning over control to a computer can have significant benefits to society as a whole, even if it causes individuals to make a slight sacrifice.

Keywords: Multiagent, dynamic simulation, traffic, human nature

1 Introduction

The purpose of this project was to give an example of a situation in which there in in fact an solution to the apparent paradox spelled out in theoretical situations such as the so-called Tragedy of the Commons. In that situation, we ae given a theoretical village with a herd of cattle owned by various individuals in the village. They have a restricted amount of commons area on which to graze the cattle. The paradox in the situation is that, unlike the traditional view of economics, the individual actions taken purely out of self interest do not help the village as a whole. If a villager chooses to increase the size of his herd of cattle, it will damage the commons, and potentially even starve the village. However, he still benefits from this overall, as he now has more cows, and is richer himself. This paradox means that people acting purely out of self interest actually hurt the group as a whole, and so the society does not succeed. We see a similar effect in the world of traffic and congestion. People will always act in their own self interest, even if it slows down the system as a whole. My goal here is to demonstrate that the paradox can be solved by having a overall intelligence which makes these decisions for the people, acting in the interest of the system as a whole, rather than the interest of a specific individual.

2 Background

Traffic dynamics are becoming an important use for agent-based modeling systems, as they provide a tangible benefits and are an excellent way to predict the behavior of a generally unpredictable system. Because a traffic system consists of multiple drivers each thinking and acting independently, the use of semi-intelligent individual driver-agents is very effective in a simulation. In this project I will be comparing this multi-agent approach (an approximation of what we see on the roads today) with a theoretically "better" approach, in which all decisions are made by a central intelligence, for the good of the system as a whole. I have done research into various traffic simulation problems, and approaches like this have been studied before, though not in the same way. In "Simulation of Traffic Systems - An Overview" by Matti Pursula at Helsinki University of technology, the history of traffic simulations is discussed, along with various ways in which it is done. I am focusing on the agent-based modeling system for this project, and I may

(time permitting) incorporate some aspects of parallel processing.

3 Structure

The program consists of four separate classes:

3.1 TrafficSim

TrafficSim.java is the main class, it incorporates the GUI of the simulation window, calculates and displays statistics in the statistic window, and keeps the other three classes organized. The simulation GUI consists of a grid of RoadSquare objects, surrounded by panels with JButtons and JSliders for various functions. These include adding to the simulation map, changing the view mode, and changing the simulation variables.

3.2 RoadSquare

A RoadSquare is an extended JButton, it stores what its "type" is, its capacity, its users, and calculates its congestion. When told what mode the TrafficSim class wants to display, the RoadSquare sets its picture or color appropriately. The view modes are: 1) Type - simply displays a picture of the road/house/factory in a calculated orientation, this is the most graphically "pretty" look for the simulation. 2) Coordinates - displays a basic color to indicate the type, sets the text of the button to the coordinates of the button (with the origin in the top left). 3) Users - similar to Coordinates in that it displays a color to shown the type, but the text is the number of Car objects currently using the road. 4) Congestion - this uses the congestion value calculated (varies with road capacity, users, simulation variables) to display a color somewhere from green to red to indicate the level of congestion. Houses are shown in white, factories in blue, empty road in gray, and unused grid locations in black. RoadSquares can be one of two types of road, a house, a factory, or an empty square. A house is defined as the starting point for a Car object, and a factory is the destination. If the RoadSquare is a house, it has a Car object, otherwise that variable is null.

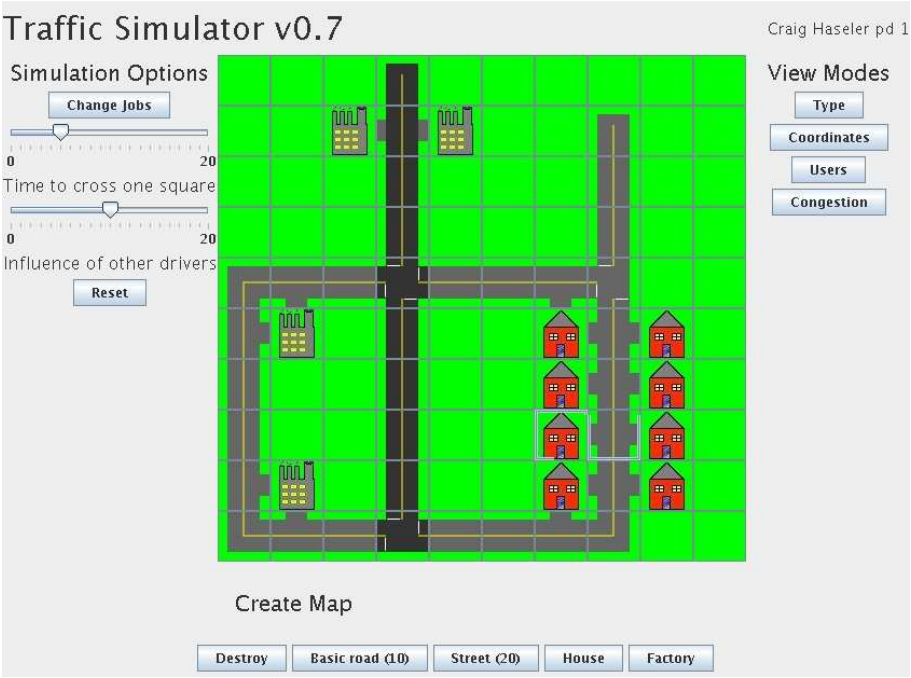

Figure 1: A screenshot of the simulation window set to view-mode "Type" with an example "map" created.

3.3 Car

A Car stores the optimal Route object from its assigned house to an accessible factory it randomly chooses.

3.4 Route

The Route class actually calculates the optimal route, currently only using the agent-based algorithm. It stores an ArrayList of RoadSquares marking the route of the Car. Together, these four classes are the backbone of the project. The project also has about eighty associated image files to make the "Type" category display correctly.

4 Evolution of Algorithms Used

In the second quarter, I focused on refining the algorithms used to do the basic sorting and searching of Routes. Rather than trying to adapt prewritten code, I chose to write my own code, as I needed a very high level of customizability. I replaced the older, very basic and very inefficient bubble sort with a Quicksort, and the old breadth-first search with a search which

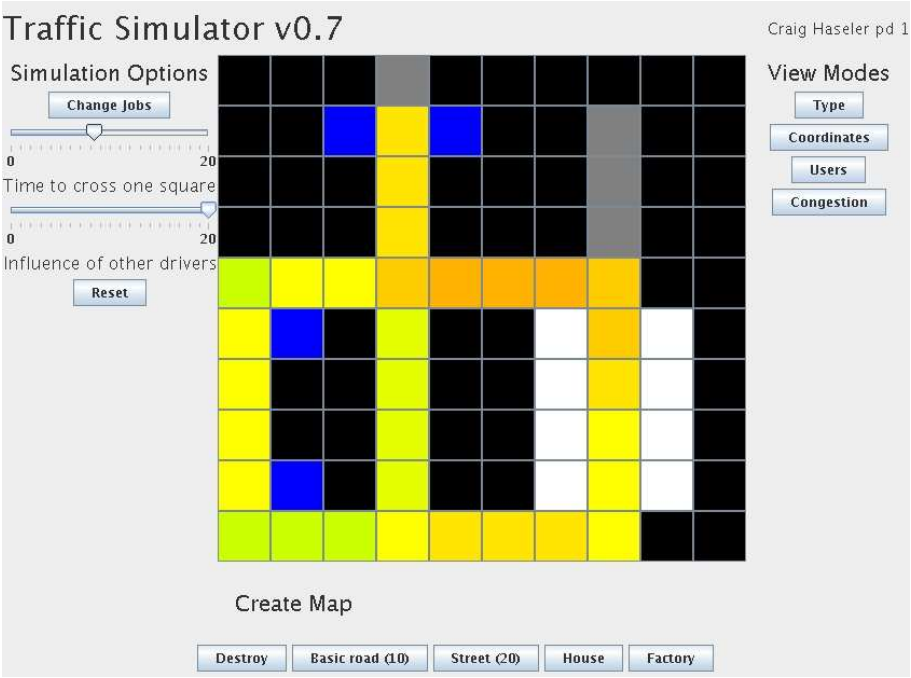

Figure 2: A screenshot of the simulation window set to view-mode "Congestion" with an example "map" created.

in this case is much more effective, an A* style search.

4.1 Quicksort

The QuickSort is a well known sorting algorithm, and one of the fastest in most cases. I chose to use it here because there is a very large amount of data to sort and an inefficient sort could slow down the program significantly. The program runs 2-3 times faster.

4.2 A Star

The A* search is one of the few search algorithms which takes into account both the distance traveled so far and the estimated distance remaining. This means it is much more efficient than other algorithms, and unlike the old breadth-first algorithm, when given a pure grid structure of nodes, will not lock up as it looks at infinite paths. As it is looking at both the distance traveled and the distance to go, it will make a beeline right for the optimal route, saving several seconds of waiting. This code change has the potential to increase the running speed of the program by up to 100x in the best case, and even in the worst case won't slow it down.

4.3 Brute Force

In the third quarter, I wrote the second of the two major algorithms used in this project. This algorithm is a simulation of a central computer controlling the cars as a group, rather than a simulation of cars making decisions on thier own. I used a simple brute force search for this, something that could certainly be refined in future. What it does is looks at every single possible combination of all cars taking each route, and looks at the statistics. It then finds the minimum combination of route for whatever the user specified criterion is (total time, average time, a combination of these, etc).

5 Results and Discussion

The brute-force based algorithm obviously takes a lot longer to run, several minutes in many cases, but often provides qualitatively better results. A key presumption behind my proposal is that in applications like this, an increase in cost of running the program is acceptable if it produces better resul ts. Moore's Law of Computing states that processor speeds will continue to increase, meaning that even if an accurate full scale simulation like mine would be too inefficient to be reliable now, it might well be fine in 10 years when we actually have an application for it. Something else important to note is that the algorithm I use in this program is completely unoptimized, consisting solely of a brute-force search. This certainly could lead to an improvement in efficiency, possibly enough to work reliable on current hardware.

6 Possible Improvements

Improvements to this project could include severel new algortihms and coding tricks, such as parallel processing, improved constraint solving, and dynamic grid size and other structural improvements.

6.1 Parallel Processing

This problem is what is known in programming as an "Embarassingly Parallel Problem". This means that it can be split up into lots of smaller problems which are independant of each other. In this case, I would have to use a tree-like structure to find the minimum - imagine a problem with 24 possible

routes and 8 parallel processors. Each processor would first find the best of 3 routes it was assigned. The next loop over it, there would be 8 routes. 4 processors would each look at 2 routes, bringing it to a totalm of 4. This would repeat twice more, and provide the answer in a much more efficient way than simply brute forcing.

6.2 Constraint Solving

The field of constraint solving deals with optimizing the brute force algorithm by throwing out routes that we can be sure will not provide the best solution. It is hard to determine how to do this, but if I could implement it, it would provide a significant improvement to run time.

6.3 Structural improvements

These improvements would add to the functionality of the program, allowing bigger grids, more types of vehicles, and different road structures.

References

[1] Matti Pursula, "Simulation of Traffic Systems - An Overview", *Journal of Geographic Information and Decision Analysis 18* pp. 1-8, 1999.

[2] Helmut Kopka and Patrick W. Daly, A Guide to LATEX, Addison-Wesley Publishing Co., Inc., 1993.

[3] Nikos Drakos and Ross Moore, LaTeX2HTML Translator Version 99.2 beta8(1.43), Macquarie University, Sydney, 1999.

[4] Walker, Janice R. et al., "The Columbia Guide to Online Style", 1995. http://www.columbia.edu/cu/cup/cgos/idx_basic.html (August 11, 2000)