Microscopic Traffic Simulation Using Genetic Algorithm

BY:

Harsh Chandra	**031202**
Nagendra Kumar	**031221**
Rahul Mishra	**031261**
Varun Chopra	**031420**

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

May 2007

Submitted in partial fulfillment of the Degree of Bachelor of Technology

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT**

CERTIFICATE

This is to certify that the work entitled, **"Microscopic Traffic Simulation Using Genetic Algorithm"** submitted by Harsh Chandra, Nagendra Kumar, Rahul Mishra and varun Chopra in partial fulfillment for the award of degree of Bachelor of Technology in 2007 of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Schandra 21.05.07

Satish Chandra
Sr. Lecturer
Department Of Computer Science & Engineering

[illegible]

Brig. S.P.Ghrera
HOD, Department of CSE & IT

Acknowledgement

We would like to express our deep sense of gratitude and heartiest thanks to our Project Guide ***Mr. Satish Chandra,*** Senior Lecturer, Jaypee University of Information Technology for guiding us throughout this project and providing us each and every resource required to make this project a success.

We would also like to thank *Mr. Sarvesh Sahare*, Software Engineer & Network analyst, L&T InfoTech, Powai, Mumbai for giving us guidelines and setting up a roadmap for development of this project.

We would also like to thank Mr. Tom Nally, Civil Engineer in New Orleans, Louisiana for providing us his code of traffic simulator developed in justbasic and also guided us through email in setting the framework of the project.

Furthermore, we would like to thank all the friends who helped us directly or indirectly in project development inclusive of typical programming involved in this project.

Lastly, we would like to show our high gratitude to the entire faculty for showing us different methodologies used in our project.

- Harsh Chandra	031202
- Nagendra Kumar	031221
- Varun Chopra	031420

Table of Contents

LIST OF FIGURES

Abstract

Optimal traffic light control is a multi-agent decision problem, for which we propose to use genetic algorithms. Our algorithm learns the expected waiting times of cars for red and green lights at each intersection, and sets the traffic lights to green for the configuration maximizing individual car gains. The experimental results show that the genetic algorithms can strongly reduce average waiting times of cars compared to three hand-designed controllers

For smooth flow of traffic in a city, the waiting time needs to be minimized. This project uses the approach of genetic algorithms for making traffic lights behave smartly. We have always found the area of artificial intelligence very interesting and has followed courses knowledge based systems, genetic algorithm and neural networks. One of the goals of this project is to gather some information about the current status in traffic simulation and to see what is possible with the current technology.

Chapter 1: Introduction

Everyone just wait waiting for a traffic light to change to green. That is why we got inspired for this project. In this project we had used genetic algorithms for making traffic lights behave smartly. We have always found the area of artificial intelligence very interesting and has followed courses knowledge based systems, genetic algorithm and neural networks.

Led by the general growth in traffic, man increasingly needs some tools to plan and control the possible evolution of the traffic network. In fact nowadays the building of any crossroad needs to be studied before in order to know what could be the best way to program the signal head. In that way traffic simulation actually has an increasing function in urbanism. But in order to make a reliable simulation, some field measurements are needed and moreover we need to compare our simulations with the real values. Moreover, in order to improve the simulation's results, a link between the different parameters of the simulation and its output is needed.

In chapter 2 there is given an overview of the problems today and some of the solution methods that has been proposed and tried.

In chapter 3 we had described some of the traffic simulators that exist.

In chapter 4 we had described the genetic algorithms and our approach.

In chapter 5 we had described our project and the technical details.

In chapter 6 we had described the limitations of our project.

Chapter 2: Traffic

2.1 The increase in traffic

In thirty years the number of car is almost tripled. In 1970 there were 184 cars per 1000 inhabitants in European countries. And in 2000 this number was up to 469. This off coarse means that the number of road users has increased dramatically. In shanghai there has been 20% increase in traffic since 1990. In Denmark there has been 30% increase in the last five years.

Government has made calculation that show that congested traffic on the main road leading into cities costs the society billion. In this calculation they have included the loss of work hours, wasted spare time and the fact that many people leave early, because they expect traffic jam.

2.2 The problems

The increase in traffic has given some problems in many cities. Traffic congestion is becoming a larger and larger problem, and specifically during rush hour it is a huge problem. Most cities are not built for the magnitude of traffic we will experience in coming years. One of the problems from this congestion is the cars stuck in traffic which are a large contribution to the pollution. Another thing is that many people than necessary for getting to and from the job. Some people actually leave for job early, due tot the fact that they expect to be caught in a traffic jam on the way.

One of the problems following from the traffic jam it can cause stress to the road users. They are caught in the situation they cannot do anything about. The people deciding the road layout have a difficult job when they are designing or modifying roads in a city. They have lots of factors to take into consideration. In can be difficult to predict how traffic would behave if you for example change the road to be a one way street only. Another thing is that they are often restricted to the current structure of the road, because it is expensive to make new roads. Especially it means when some building has to be removes as it was the case in New Delhi when metro train project was started. It is also difficult to change things in the middle of the city, because you cannot just close the roads while you make the alteration.

2.3 Spotting the problems

There are several ways to check how much traffic there is at a given place. In Australia there are acoustics sensors to determine how much cars that are waiting for a traffic light to turn green. Using this information the traffic lights decides what to do.

In FINLAND a firm called Finnra are using mobile phones to see where the traffic is situated. Today 80% of the Finland population owns a mobile phone. Every one in a while a mobile phone swaps information with the base station, and this information can be used to track the route of the mobile phone and its owner. By measuring the time a phone is connected to a base station they think it will be possible to decide the travel time of the car. Comparing this travel time with the previous measurements we can tell whether the car is moving fast or slow. They think that it will be enough to monitor 5% of the phone to gather the information about the traffic pattern.

In BERLIN, they are trying something very different. Here they are taking infrared pictures from the altitude of 4500 meters. The moving cars can be spotted with their heat signatures. An advantage with the infrared pictures over normal pictures is that it is easier to spot the moving cars (parked cars are cold) and also clouds and bad weather is not a problem.

As cars are more and more advanced, they use GLOBAL POSITIONING SYSTEM (GPS). There precise information from a road user would be perfect to track information about the traffic. But we might have to wait for some more years for GPS to be standard in all cars.

2.4 Some solution methods

What can we do to solve these problems? One obvious solution is to build roads with multiple lanes to increase the capacity. But in many cities it is not possible due to the placements of buildings. It can also be a costly affair to test new strategies on real traffic.

It could also be a solution to make some restriction to the traffic in the center of the city. This could be done by introducing road tax on some of the most congested roads or remove many if the possibility of parking in the center of the city. These solutions are not popular among the road users so in most cases it is not the real option.

2.4.1 Green Waves

The idea behind the green waves is to let the traffic lights on certain stretches of road be times so you do not have to wait for any red lights on this stretch. A problem with this approach is that it is difficult to time traffic lights so the green wave works in both directions. Another problem is that this is the fixed timing of the traffic lights so if the average speed of the road user drops (may be due to bad weather), green waves will not work.

2.4.2 Public Transportation

A way to minimize the traffic could be to use more public transportation like busses. The problem is that it often takes much longer time to travel than it would if you use your own car. But giving special lanes to bus can minimize this time. But it is not possible to have bus lanes everywhere. Simply, there is not enough space for them.

It seems that bus is not that influenced with the traffic delay, because in comparison the bus is slower, and has to make many stops. But a delay of 2 minutes for a bus can make a big difference for many passengers as they might have to catch another bus or train. This means that a delay of 2 minutes for a bus can end up in a delay of 15 minutes or more as passengers have to wait to catch another bus. This means many people earlier than actually necessary to be sure that they arrive in time.

2.4.3 The Cars

The cars of today are much safer than they were twenty years ago. They are lot more comfortable and quiet. These factors are some of the reason to why people are driving faster and faster. Companies like Mercedes, Audic , Lexus are equipping there luxury cars with *adaptive cruise control* that controls the length and slow down the cars if it is to close to the in front of it. This feature should help in lowering the number of accidents, especially multiple collisions, which often are caused by the fact that people do not keep enough distance to the car in front.

2.4.4 Electronic Road Signs

There has been suggested the idea of electronic road sign (ERS) which help to regulate the traffic according to the given condition.

E.g.: In bad weather the speed limit could be lowered. Also, if it looks like there is an upcoming traffic jam, the traffic from behind could be instructed to lower the speed. It is likely that people would take the speed limit that is based on actual condition more seriously than normal speed limits.

By simulation, it has been shown that by imposing speed limits it is possible to increase the flow of traffic during rush hours. This is because the difference of speed is reduced. At all other times than rush hours it has a negative effect. So, though it sounds odd , it might be a good idea to reduce the speed of the car during rush hours.

The ERS could also be used to inform the road users about the current traffic situation. This could be to tell that there has happened an accident up ahead or that the road is slippery.
A simple version of ERS has been used in HOLLAND and GERMANY for over 20 years now, and is estimated to have reduced the number of people killed by 25%-35%. It has also been proposed to be used in Denmark, but was according to the department of justice to expensive.

2.4.5 Traffic simulation

Today it is possible to make situation on how a change in the current road layout would affect the traffic. A thing like micro simulation is known to predict events that are not explicitly modeled. Therefore it can be used for testing roads layouts, that otherwise would not have been tested.

2.4.6 Traffic Lights

Another way of solving (or at least minimizing) some of the problems is to make the traffic lights more effective. By effective we mean that they should minimize the car waiting time.

Chapter 3 : Simulators

There exist many simulators. In this chapter we had tried to give an overview of some of them. Some are small projects, and others are full scale models used to simulate the traffic.

3.1 SuRGE

SuRGE (Swarms under R&J using evolution) is *a swarm-based* traffic simulation system with evolution. It is developed by Ricardo Hoar & Joanne Penner. The key idea in their simulators is to let car behave as social insects, specifically ANTS. When the cars are driving around in the environment they leave a trail of pheromones which fade over time. This trail can be detected by all other cars and is used for the coordination.

The simulator is based on a two-dimensional user interface where the swarm of car can navigate on roads, connected by various type of intersection. The cars obey traffic laws and avoid collision with other cars. At the intersection there are traffic lights with timing sequence. It is timing sequences that they use evolution on. They have introduced what they call swarm voting to increase the speed of adaptation. For each time step that a car have to wait for a specific traffic light, it gives traffic light a vote. This can be used to indicate which traffic light that behaves worst and therefore needs an evolution. In their test cases the swarm voting actually helps. They had made some test on normal traffic, rush hour traffic and a special case, where they make some nice improvement.

We can see a possible problem with this approach. Although the waiting time is reduced but it can't stop blocking the traffic forever. Another thing is that they have only made test on simple examples. The cars in their simulation are quite simple. But we really think that the main idea was nice.

3.2 Paramics

Paramics is an acronym derived from Microscopic simulation ON parallel computers. It is commercially available simulator developed by SIAS (transportation engineers) and QUADSTONE (specialist in high performance software). Actually Paramics consist of three parts: modeler, analyzer and processor. The Paramics modeler is the part where you build the model, and have a nice GUI showing the traffic. The Paramics analyzer is used to analyze data from simulation and compare it with real world simulation. Last there is Paramics processor used for simulating in batch mode, without the GUI. Without having to visualize the data this part is much faster at doing simulation. Off course this can only be done when you have the final model. In Paramics they have included such things like public traffic and bus priority.

3.2.1 Microsimulation

In some articles there is an overview of Paramics microsimulation and some real world application of the simulator. They claim that they have designed the simulator from scratch to take advantage of modern architecture. Many other (Older) systems are written in a general high level language. The development in the computer industry had provided us with much computing power than we had 10 or 20 years ago. This development had made possible to model all aspects of traffic system. This means that it is possible to go from a *pipe flow analogy to microsimulation*. In a pipe flow model a driver is going from home to work, is being represented as being everywhere along the route in the period it takes to go to work. In microsimulation the driver is modeled individually and only happens at one place at a time. Because of individually modeling of each driver, there is also a need of great computing power. We think that this is the main reason to why microsimulation first recently had been applied to traffic simulation in real world scale.

Almost all models/simulators ignore real world things and use an idealized view of the world-some more than others. By going from a pipe flow analogy to microsimulation we get a lot closer to the real world.
Also an accident can cause a traffic jam. It would be difficult to model an accident in an pipe flow model, and when it is done, how can you see that it is an accident that caused the traffic jam? And if there is a traffic jam, what caused it? Too much traffic? A faulty coding of a traffic light? Or something completely different? In a microsimulation model you can directly see the driver and the impact on the overall traffic situation. One of the problems with simulator is that small coding errors can give large simulation error. Therefore it is extremely important to be able to validate your model.

3.2.2 Vehicle Model

The movement of individual vehicles in the Paramics simulator is controlled by three interacting models representing vehicle following, gap acceptance, and lane changing. The interaction of these three models applied directly to each vehicle is the basis for the simulation behavior. In the model there is two main driver characteristics: *aggression* and *awareness*. Some of the parameters influenced by these characteristics are: gap acceptance, acceleration, top speed, headway and propensity to change lane.

Each vehicle has a unique perception of travel costs and choose to route to its destination, that minimize time and distance. In each node it dynamically re-evaluates its route to see if there is a better route. This is very useful because traffic is highly dynamical. But in our opinion it is also a king of cheating: Do human drivers actually alter their route along the way? Personally we only choose other route if the route we are currently following is blocked. This way of simulation ensures that if a road get blocked or heavily loaded with traffic, the driver will try to find some other route. We are not convinced that this is the case in real life.

3.2.3 *Real World Experience*

The first major urban area in the world which was represented by a microscopic traffic simulation by Paisley in SCOTLAND which had a population of 80,000. Earlier there were tried, other transportation studies, but they had not provided sufficient information, so in 1996 they transformed their data for use in Paramics simulator. Since then Paramics simulator has been used in many other places.

By having done many real world projects in real life, they had discovered some funny things and got some special ideas. One of the things that they have observed using the Paramics simulator is that incidents on a highway can an hour after it has been cleared; still have effects on traffic 5 km away. This can explain some of the peculiarities that can be observed in the traffic.
One extremely nice idea (in our opinion) , is that to have *DISPLACED RIGHT TURN (see **figure 1**)*

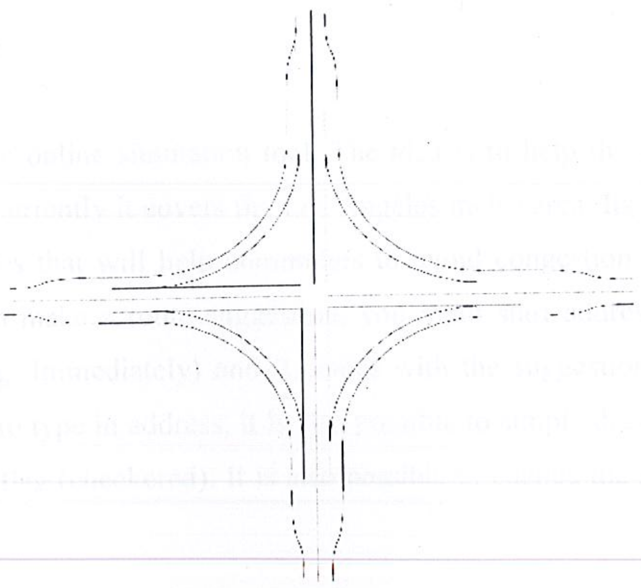


Figure 1 : Displaced right turn

DISPLACED RIGHT TURN is the idea to make storage capacity for drivers going straight and turning to the left, so it is possible to make right turn. One major problem with this is that it needs a lot of space. It is impossible to imagine a layout like this in certain places like *cannaught place, New Delhi.*

3.2.4 The Simulator

We had downloaded an evaluation copy of the Paramics simulator. After having tried it out, it is very clear that it is highly advanced simulator. The GUI is quite nice and there is no doubt that the function in the simulator is highly professional. As it can been seen in figure 1 it is possible to take a 3D view of the map. The view can be modified by using the mouse and the keyboard. It is possible to make 10 predefined views which are accessible from a menu in the main menu. All we must admit that the Paramics simulator is very impressive.

3.3 Traffic Dodger

Traffic dodger is an online simulation tool. The idea is to help the commuters manage their travel plans. Currently it covers the Los Angeles metro area. Its main feature is that it can suggest routes that will help commuters to avoid congestion. This page is quite intuitive to use. To make a route suggestion, you input start address, end address and when to travel (E.g. Immediately) and it comes with the suggestion. If you think it is quite cumbersome to type in address, it is also possible to simple drag drop and start flag (green) and a goal flag (checkered). It is also possible to change the size of the map and the zoom in factor.

One of the very nice things about the traffic dodger is that it contains real life data. And here we not were talking about data from last year or yesterday, but the most recent data. They get speed data from CalTRANS (California department of transportation) via PeMS

(Performance measurement system). These data are time stamped and can have a delay of up to 10 minutes. Some of this delay they are physically trying to remove in cooperation with CalTRANS. But there will be some delay from the data is gathered until it can be provided to the end user. One way they are trying to work around this, is by trying to predict the correct values. This is done by using a given sensors historical value. This is also used when the sensors is failing (no data returned).

In addition to the data in CalTRANS, they are also receiving data from CHP (California Highway Patrol)

3.4 City Traffic

City-traffic is a German traffic simulator. We have tried to contact the people behind the simulator via email but unfortunately they didn't reply so the only material we have about this simulator is an 'article' from their website.

City Traffic is an agent based micro model of traffic. The system is based on actual traffic information received from various sensors in real time and a microsimulation utilizing real traffic data. The person behind city traffic has pointed out the progress in three areas that has made it possible to make simulation as accurate as they are doing.

3.5 Traffsim

This was the well written open source code developed in justbasic. The data followed in this simulator was very close to the real world. The cars even stop at instance for fuels to be refilled.

Chapter 4 : Genetic Algorithm

We have chosen to use genetic algorithm (GA) as the main idea behind our algorithms. Therefore we will give a short motivation for and introduction to the area of GA. We have omitted many details and aspects of GA. This chapter is only included to give the reader an idea of what GA is. That's why we had left many proofs and details about the statement we had provided.

4.1 *The Main Idea*

So how do we make the computer able to learn from interacting with the environment? This is where the field of AI is useful. It is off course an idealized way of learning because we are somewhat limited to the computer and the programming language.

Genetic Algorithms (GA) are search algorithms that simulate the process of natural selection and survival of the fittest. GA attempt to find a good solution to some problem (e.g., finding the global maximum of a function) by randomly generating a collection of potential solutions to the problem and then manipulating those solutions using genetic operators. In GA terminology, we say that we generate a population of solutions and refer to each solution as an individual. Each solution gets a scalar *fitness* value which is a numerical assessment of how well it solves the problem. The key idea is to select for reproduction the solutions with higher *fitness* and apply the genetic operators to them to generate new solutions. Again, in GA terminology these new solutions are called newborn individuals. Through mutation and re-combination (crossover) operations, better newborn solutions are hopefully generated out of the current set of potential solutions. This process continues until some termination condition is met. In the case of numerical optimization, each solution is a vector of numbers (real or integer). The GA attempts to find the vector with maximum fitness in a specified hypercube1.

4.2. Genetic operators

4.2.1. Selection Operator

- **Fitness-based selection**

Each individual's probability of being selected depends on its fitness value. The most common selections of this type are:

i. Roulette wheel selection:

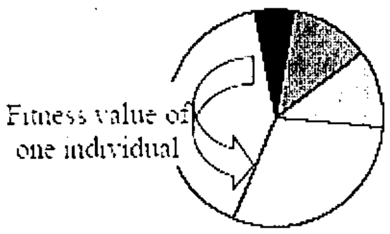


Figure 4.1: Probability example in a roulette wheel selection

Each individual's probability of being selected is proportional to its fitness value. The individuals are mapped to contiguous segments of a line, so that each individual's segment is equal to its fitness. A random number is generated and the individual whose segment spans the random number is selected.

ii. Stochastic universal sampling:

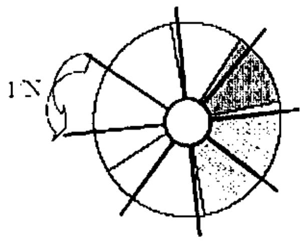


Figure 4.2: Example of a selection based upon stochastic universal sampling

Exactly like in the roulette-wheel selection, the individuals are mapped to contiguous segments of a line. Considering N the number of individuals to be selected, then the distance between the pointers is 1/N and the position of the first pointer is given by a randomly generated number in the range [0, 1/N]. Thus all the N individuals are chosen with only one random sample.

4.2.2 Crossover operators

i. Point crossover:

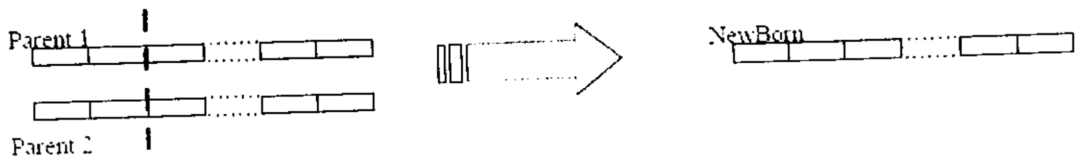


Figure 4.3 : Point Crossover

The point crossover operator aligns the genotypes of the parents. A crossover position is then randomly selected with uniform probability and the part of the first parent's genotype before the crossover position is copied on the corresponding part of the newborn. The other characteristic of the newborn comes from its corresponding place in the second parent's genotype.

ii. Linear crossover:

The linear crossover operator is specific to the floating point representation. It produces a newborn whose components are convex combinations of the corresponding components in its parents.

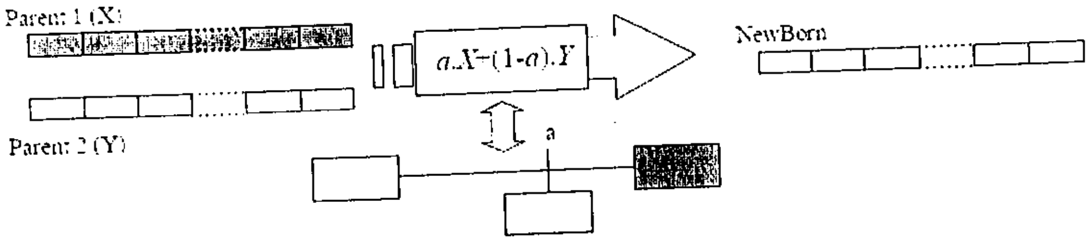


Figure 4.4: Linear crossover

For example as shown in *figure 1.6*, if X represents the first parent and Y the second parent then the newborn is $a.X+(1-a).Y$ where a is a random value selected uniformly in the interval [0,1] (with a=0.5, linear crossover degenerates to arithmetic crossover), this is the line-type one. If for each gene the same formula is applied with a new choice of a, we obtain its spacetype form, because in the first case the newborn belongs to the segment XY, and in the second case it belongs to the hypercube formed by its parents.

iii. Random crossover:

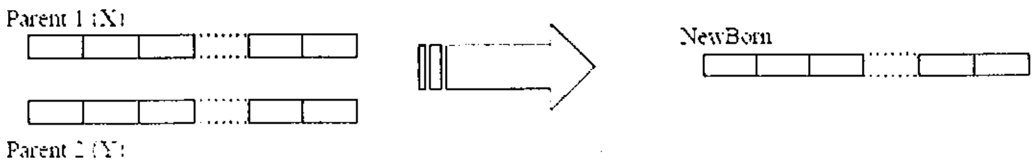


Figure 4.5: Random crossover

The random crossover operator can be applied to any type of representation. If applied to the floating point representation, each vector component of the newborn is selected randomly (with equal probability) from either parent. In the case of binary representation, each bit in the newborn is selected randomly from the corresponding position in either parent. This operator introduces a lot of variability (diversity).

4.2.3 Mutation operators

i. Uniform mutation:

Uniform mutation replaces each component of a solution vector with a random value uniformly selected from the component range.

ii. Non-uniform mutation:

Non-uniform mutation takes the stage of optimization into consideration. At the beginning of the optimization it acts just like uniform mutation. It then becomes more and more conservative about the amount of change it makes to a vector component as the optimization progresses.

Let the component's value be x with lower bound l and upper bound u and assuming the search is at iteration t and the maximum number of iterations is T, then the mutant value is

$$X_{new} = x - (u - x).r.(1 - \frac{t}{T}).scale \qquad \text{With probability 0.5}$$

or

$$X_{new} = x - (x - l).r.(1 - \frac{t}{T}).scale \qquad \text{With probability 0.5}$$

Where r is a random value selected uniformly in the interval [0,1] and *scale* is a number between 0 and 1 that decides how conservative the mutation should be.

4.3 The global architecture of our genetic algorithm

Our GA has been programmed as the skeleton of any GA on which the user should be able to use all the coded methods he wants, in fact all the methods described in appendix A except the point crossover. We will present here this skeleton, the meanings of the different parameters the user can choose in the interface as well as their default value.

4.3.1 The main algorithm

Our main algorithm looks like most of the GA except that it has to be used with lots of. Its skeleton is described below and a view of it is shown in ***figure 4***. Firstly the size of the population is given by the user through the interface, and the number of genes is extracted from the number of parameters we need to calibrate. In fact our genes have real values between 0 and 1; so our algorithm can be applied for any optimization purpose with a proper evaluation function.

To begin with, the initial population is randomly chosen or it is imposed by the user. An initialization step has been created because the selections require the fitness of the population and the replacement strategy needs the evaluation of both the parent and the children.

Then two parents are selected and form a newborn with an appropriate crossover. These two steps are repeated in order to have a newborn population with the same amount of individuals.

Some mutations are added to the newborns, with a probability fixed by the user (the default value is set to 0.5), and then the children's population is evaluated. So with this population and the old one, a replacement method is used to form the new population which will be put back in the algorithm if the final conditions are not satisfied. These conditions are the number of generations and the minimum error we try to reach; they are both fixed by the user.

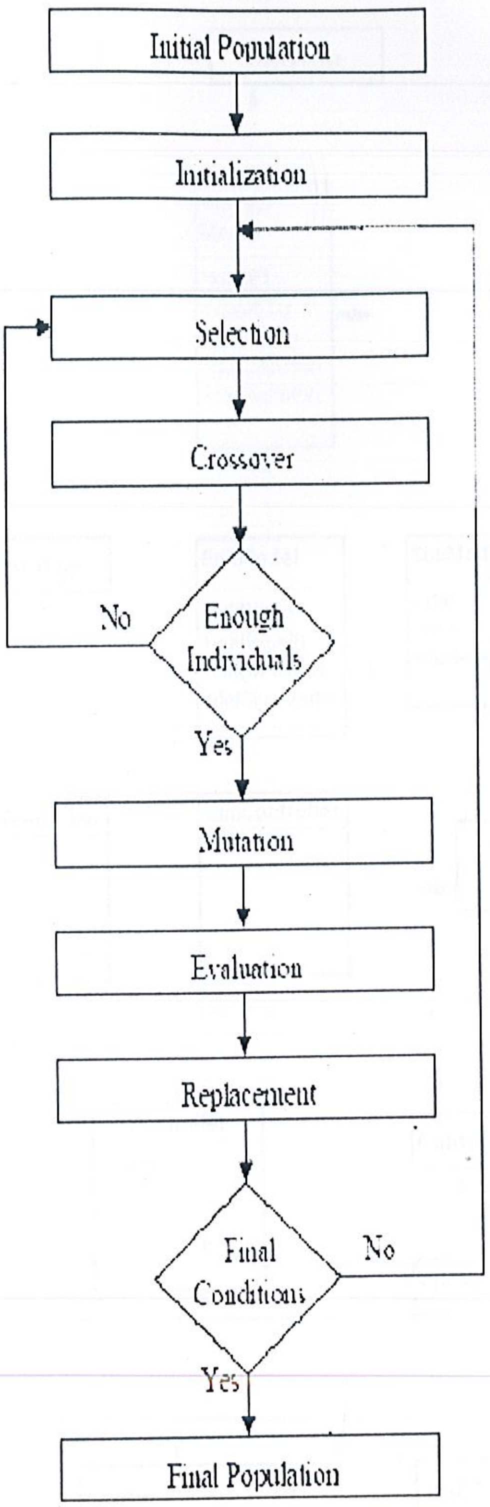
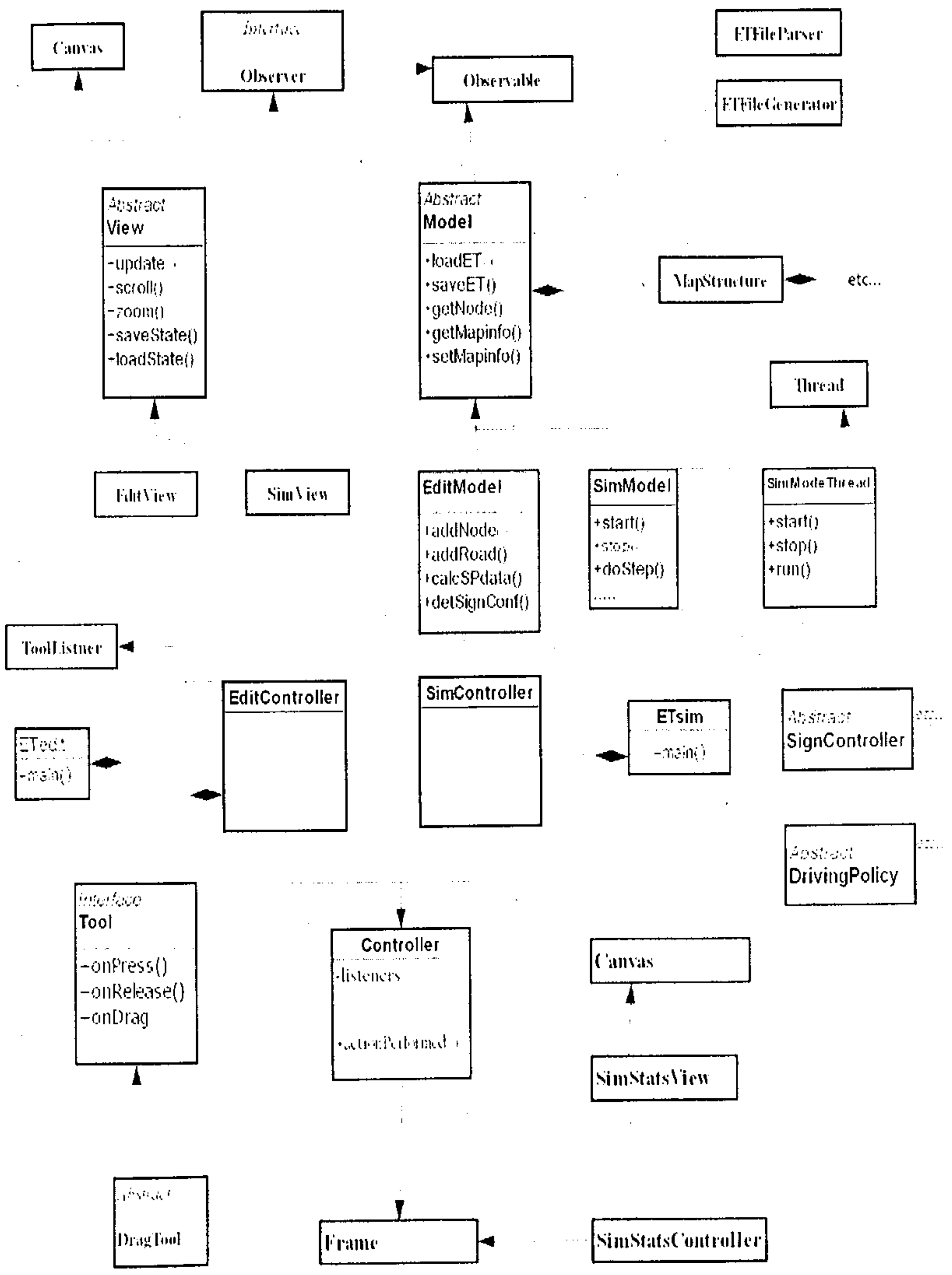
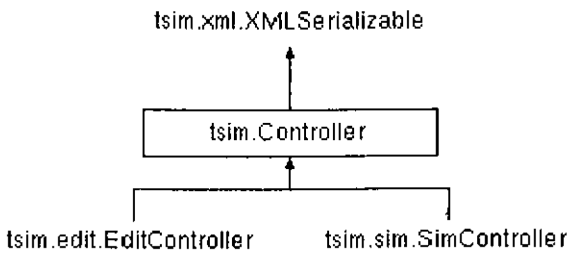


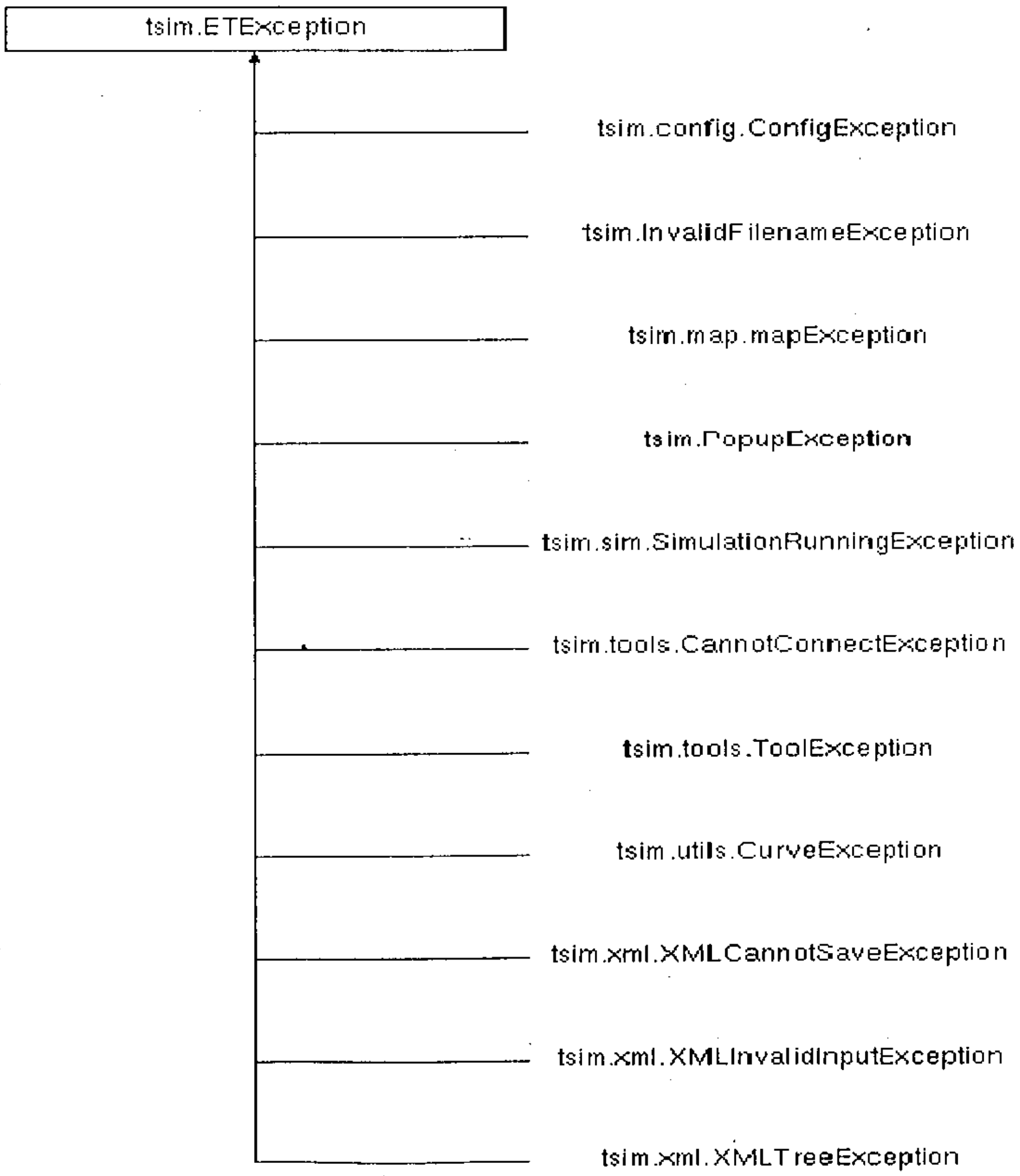
Figure 4 : The Key Concept Of genetic algorithm

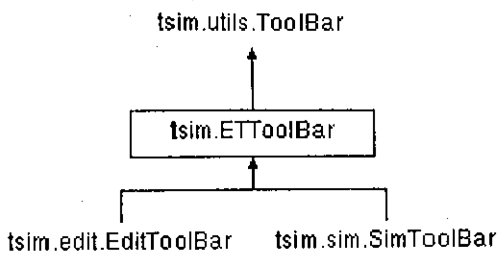
4.3.2 The UML Model

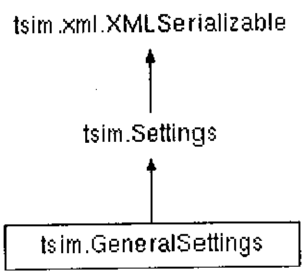


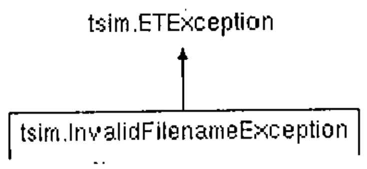
4.3.3 The Class Hierarchies

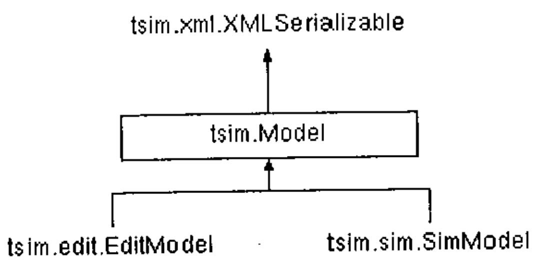


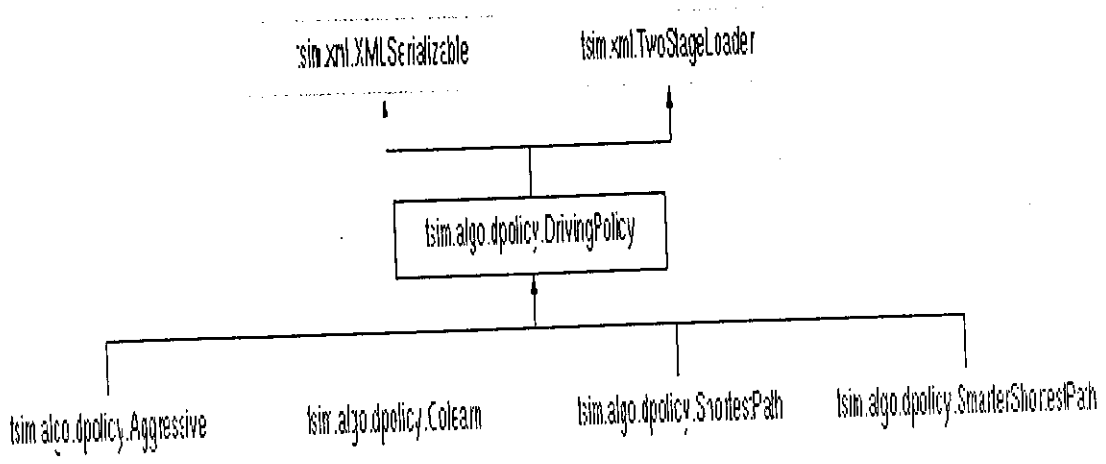

tsim.ETException
tsim.config.ConfigException
tsim.InvalidFilenameException
tsim.map.mapException
tsim.PopupException
tsim.sim.SimulationRunningException
tsim.tools.CannotConnectException
tsim.tools.ToolException
tsim.utils.CurveException
tsim.xml.XMLCannotSaveException
tsim.xml.XMLInvalidInputException
tsim.xml.XMLTreeException

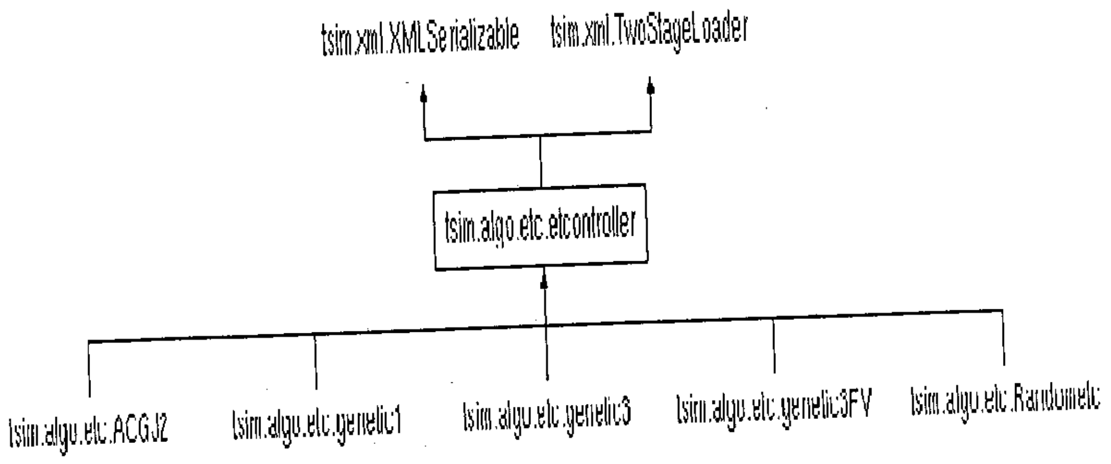

tsim.utils.ToolBar
tsim.ETToolBar
tsim.edit.EditToolBar
tsim.sim.SimToolBar

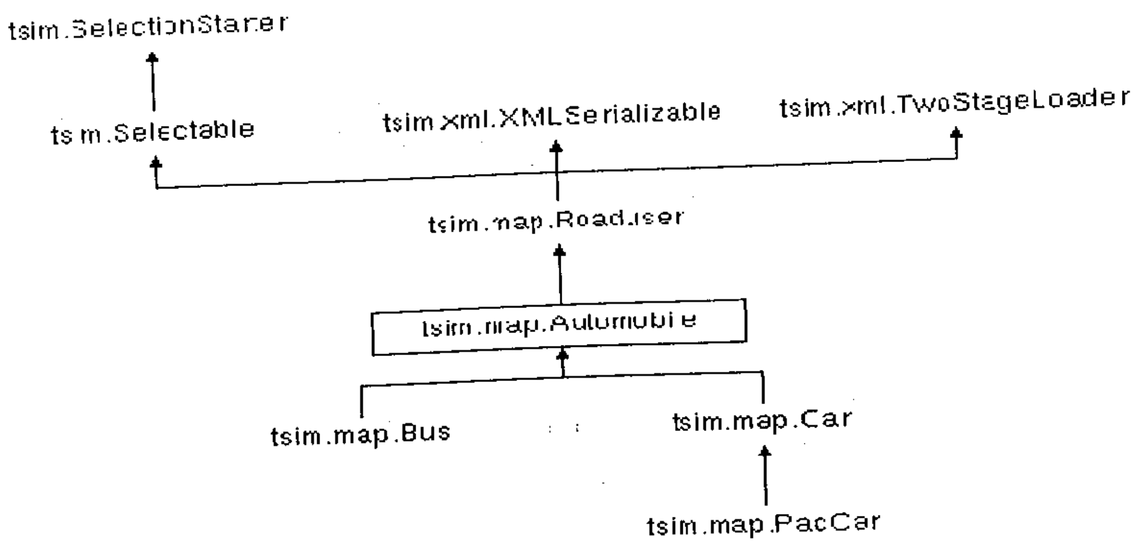

tsim.xml.XMLSerializable
tsim.Settings
tsim.GeneralSettings

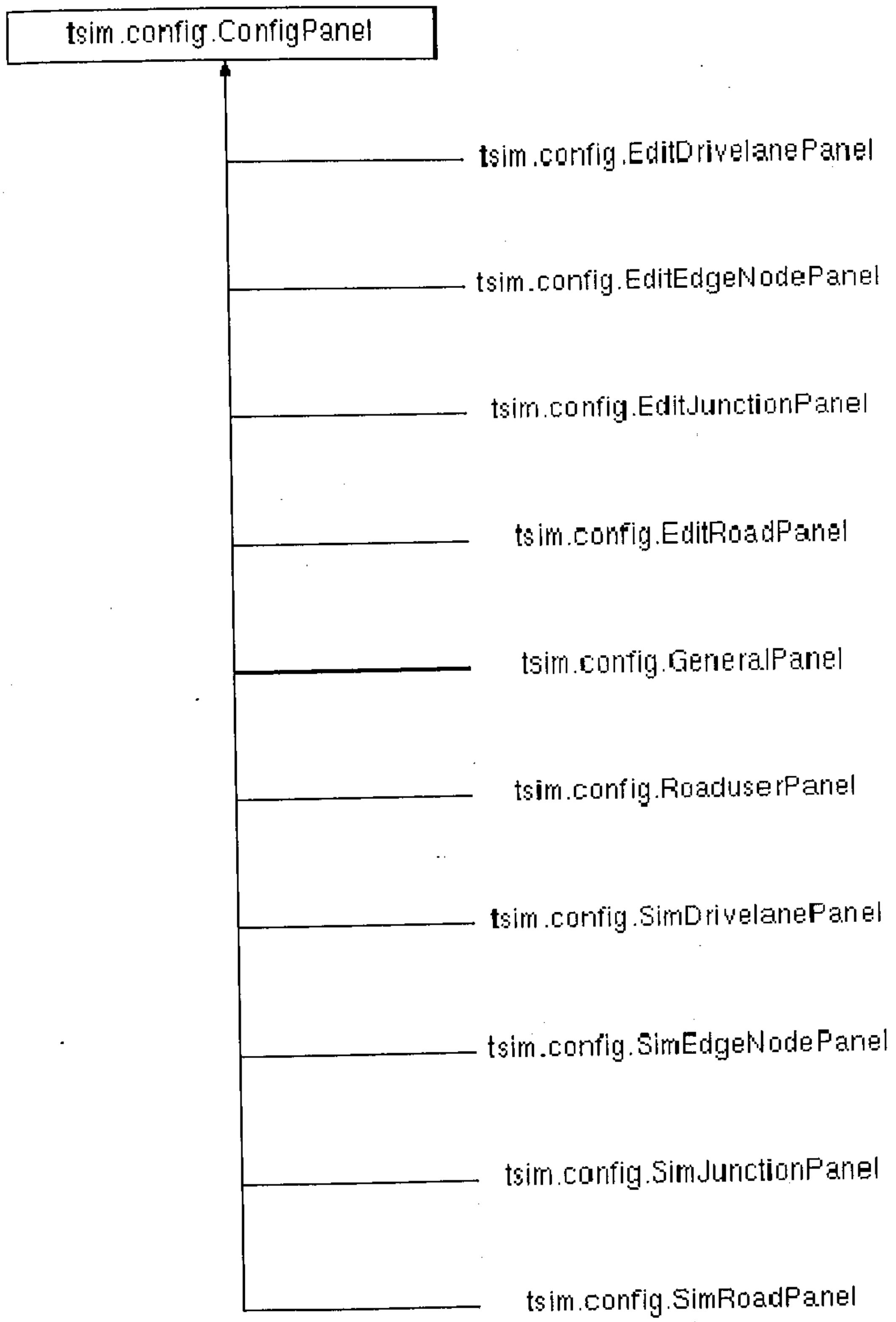

tsim.ETException
tsim.InvalidFilenameException

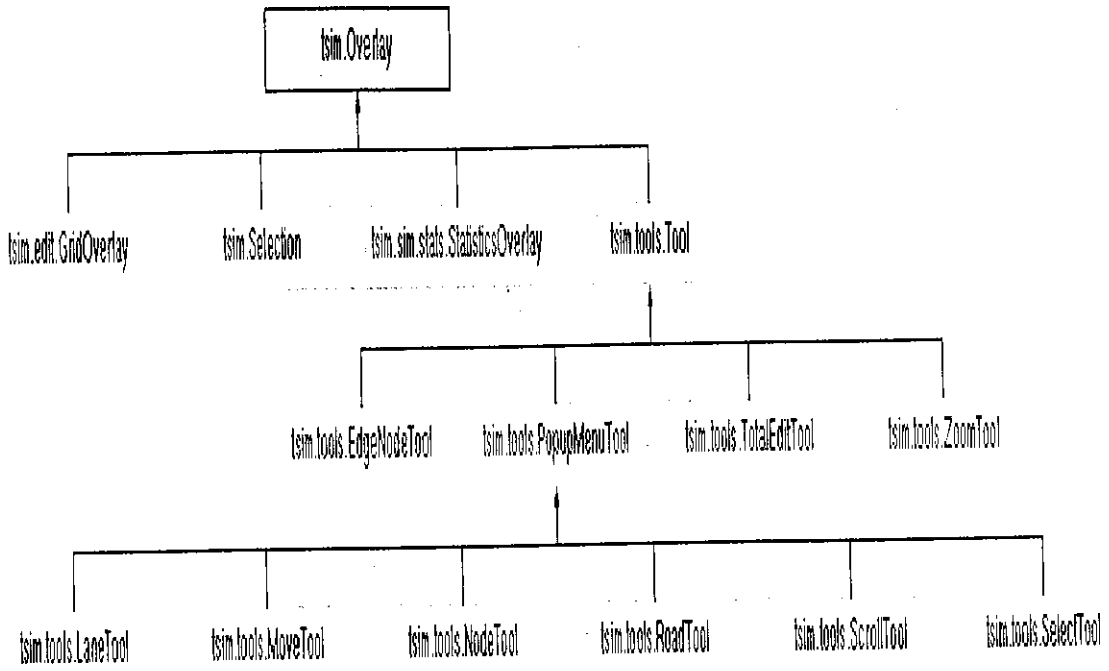

tsim.xml.XMLSerializable
tsim.Model
tsim.edit.EditModel
tsim.sim.SimModel

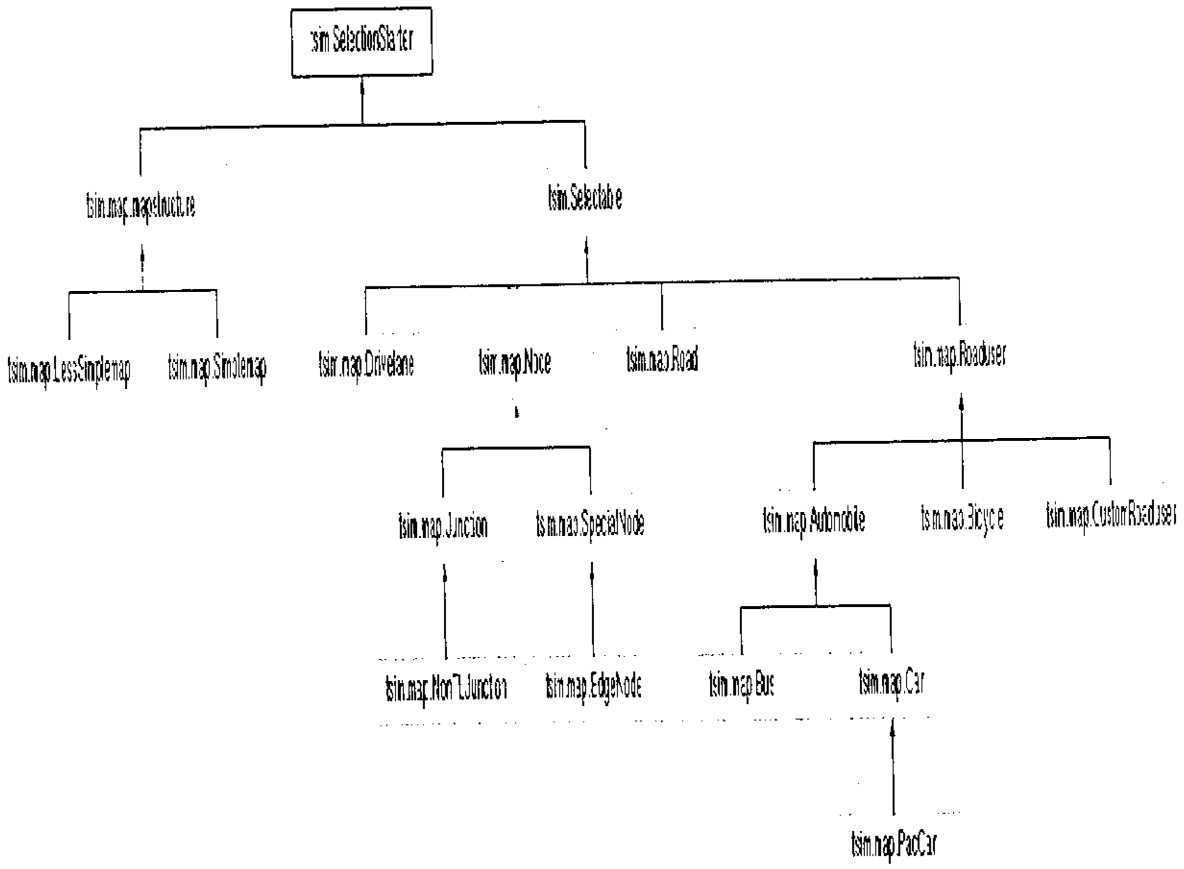

tsim.xml.XMLSerializable
tsim.xml.TwoStageLoader
tsim.algo.dpolicy.DrivingPolicy
tsim.algo.dpolicy.Aggressive
tsim.algo.dpolicy.Coleam
tsim.algo.dpolicy.ShortesPath
tsim.algo.dpolicy.SmarterShortestPath


tsim.xml.XMLSerializable
tsim.xml.TwoStageLoader
tsim.algo.etc.etcontroller
tsim.algo.etc.ACGJ2
tsim.algo.etc.genetic1
tsim.algo.etc.genetic3
tsim.algo.etc.genetic3FV
tsim.algo.etc.Randometc

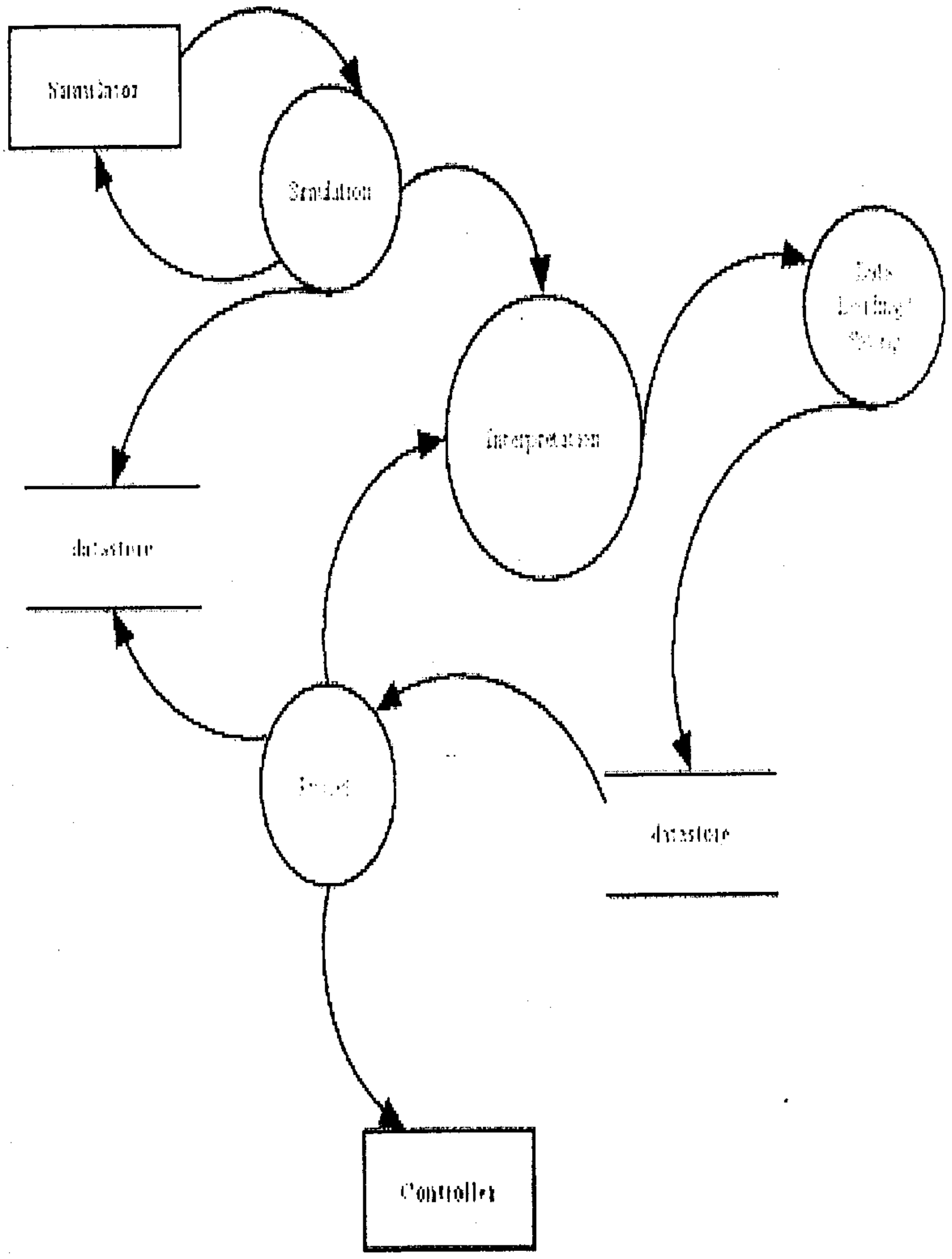

tsim.SelectionStarter
tsim.Selectable
tsim.xml.XMLSerializable
tsim.xml.TwoStageLoader
tsim.map.RoadUser
tsim.map.Automobile
tsim.map.Bus
tsim.map.Car
tsim.map.PacCar

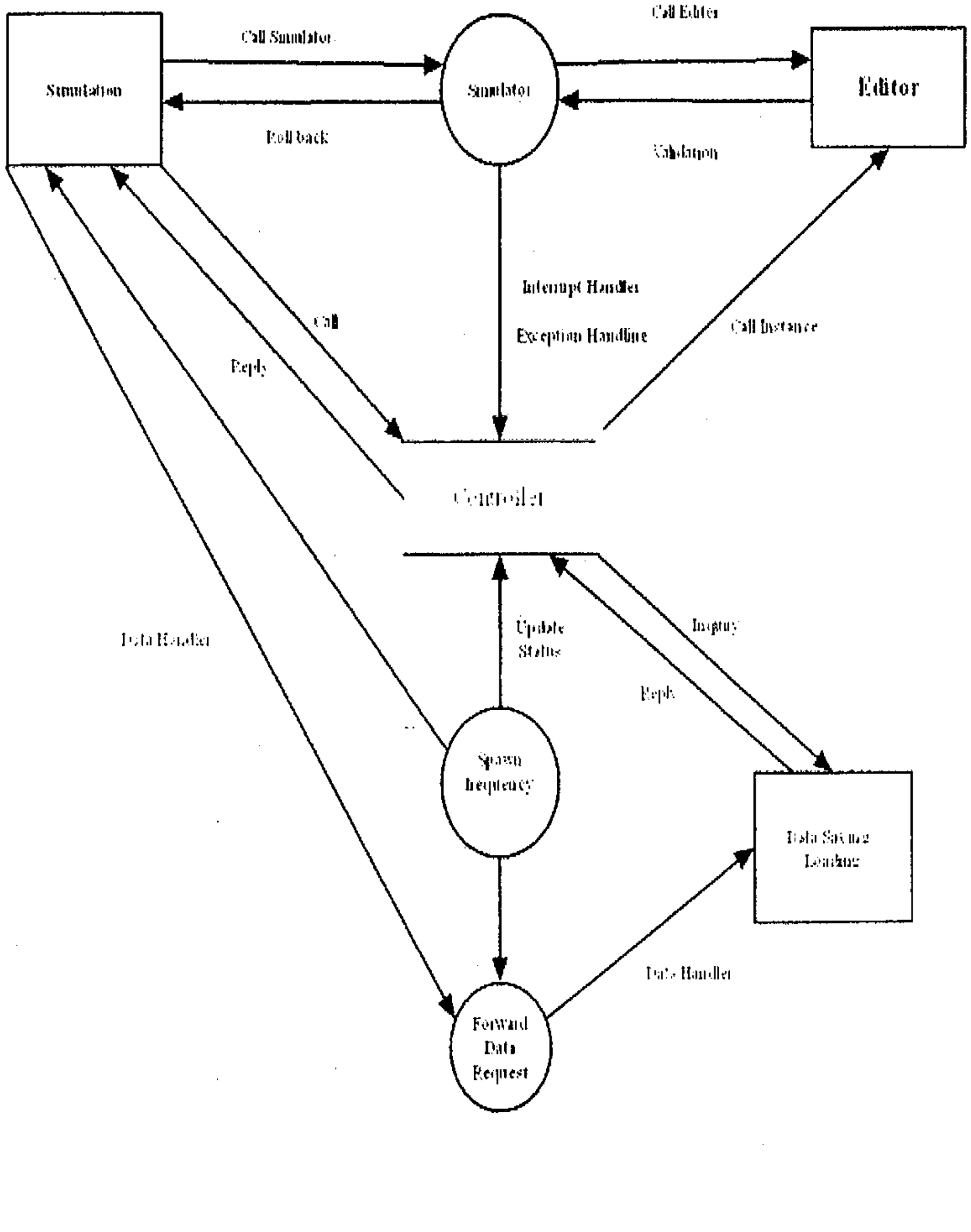

tsim.config.ConfigPanel
tsim.config.EditDrivelanePanel
tsim.config.EditEdgeNodePanel
tsim.config.EditJunctionPanel
tsim.config.EditRoadPanel
tsim.config.GeneralPanel
tsim.config.RoaduserPanel
tsim.config.SimDrivelanePanel
tsim.config.SimEdgeNodePanel
tsim.config.SimJunctionPanel
tsim.config.SimRoadPanel


tsim.Overlay
tsim.edit.GridOverlay
tsim.Selection
tsim.sim.stats.StatisticsOverlay
tsim.tools.Tool
tsim.tools.EdgeNodeTool
tsim.tools.PopupMenuTool
tsim.tools.TotalEditTool
tsim.tools.ZoomTool
tsim.tools.LaneTool
tsim.tools.MoveTool
tsim.tools.NodeTool
tsim.tools.RoadTool
tsim.tools.ScrollTool
tsim.tools.SelectTool


tsim.SelectionStarter
tsim.map.mapstructure
tsim.Selectable
tsim.map.LessSimplemap
tsim.map.Simplemap
tsim.map.Drivelane
tsim.map.Node
tsim.map.Road
tsim.map.Roaduser
tsim.map.Junction
tsim.map.SpecialNode
tsim.map.Automobile
tsim.map.Bicycle
tsim.map.CustomRoaduser
tsim.map.NonTLJunction
tsim.map.EdgeNode
tsim.map.Bus
tsim.map.Car
tsim.map.PacCar

4.3.4 Data Flow Diagram




Call Simulator
Simulation
Simulator
Call Editor
Editor
Roll back
Validation
Interrupt Handler
Exception Handling
Call
Reply
Call Instance
Controller
Update
Status
Inquiry
Reply
Data Handler
Spawn
Frequency
Data Saving
Loading
Data Handler
Forward
Data
Request

Chapter 5 : Easy Traffic

5.1 Introduction

We would like to start this chapter with a quote by Stephen Druitt

"If a microsimulation model doesn't look right, then it's probably isn't and vice versa. It is little more than applied common sense"

This quote quite nicely sums up why microsimulation has became the standard way of doing simulation. Much of our own experience has proven to us that microsimulation is a good way of doing simulation on the computer.

5.2 The Editor

The Editor is the place to start exploring easy traffic creative capabilities. The Editor window is the part of the program in which maps can be created, saved, loaded, and modified.

Map creation is the process of placing nodes, and linking them with roads to form a valid map structure. Afterwards, the number of lanes on roads, algorithms for the traffic lights and starting frequencies for road users can be set. Road users and traffic light changes are not visible in the Editor, they can be visible only in the Simulator.

When we run the software, it starts with the simulator and then we can switch over to editor for map creation or editing. Following is the screenshot of how an editor looks and how we can open the editor from the simulator window.

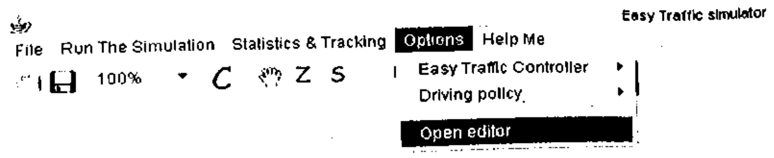


Figure 5.1 : Screenshot for opening editor from the simulator window

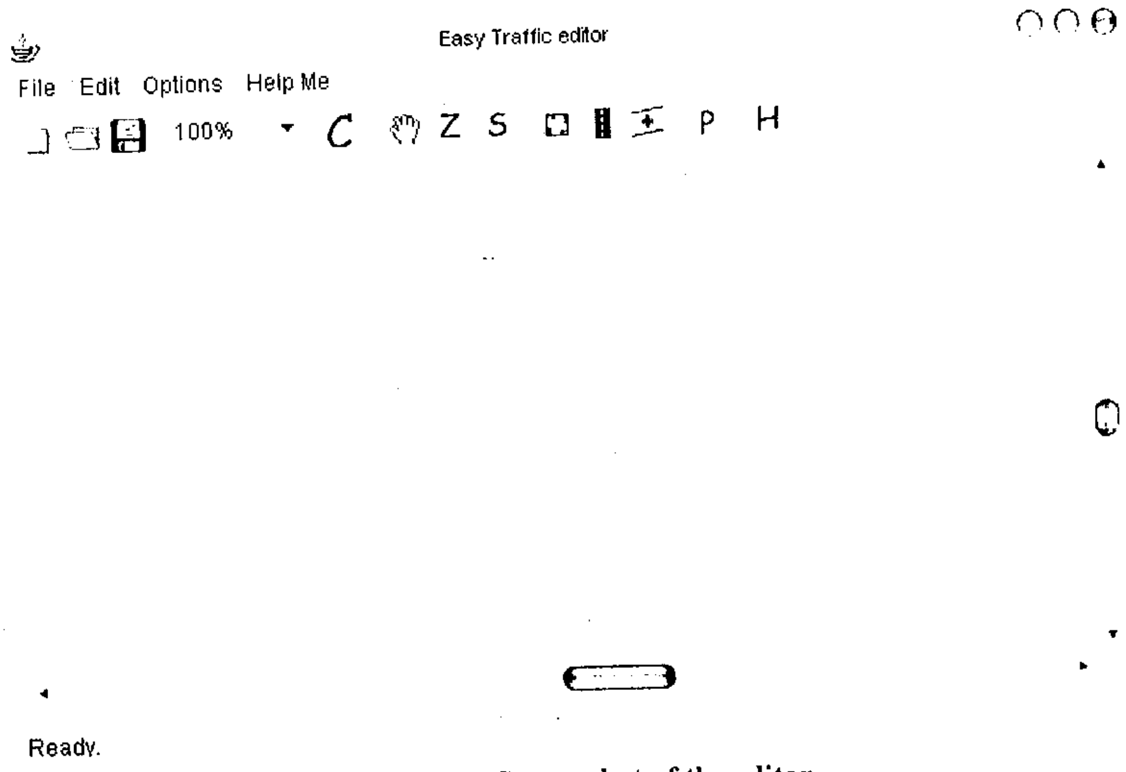


Figure 5.2: Screenshot of the editor.

5.2.1 The Editor Interface

The editor consists of the following menus:

FILE

• New	: for creating a map file from scratch.
• Open	: for opening a map file.
• Save	: for saving a map.
• Properties	: for editing additional info on a map.
• Quit	: for quitting Easy Traffic.

EDIT

• Delete	: remove selected portion of infrastructure
• Select All	: select all Roads and Nodes
• Deselect	: reset any selection

OPTIONS

• Toggle Grid	: switch grid overlay for reference on
• Change Size	: change the size of the map
• Validate	: check all couplings between Roads and Nodes
• Settings	: General Settings

HELP ME

• Help	: aid for the common user
• Technical Details	: precise guide to the easy traffic code
• Website	:opens the easy traffic website in a browser
• About	:a little note about easy traffic

5.2.2 The Editor Toolbar

The Editor's Toolbar contains the following:

- **The New button**
- **The Load button**
- **The Save button**
- **The Zoom Box -**

displays current zoom factor and allows adjustment to values:

  - 25%
  - 50%
  - 75%
  - 100%
  - 150%
  - 200%
  - 250%

• **The Center button**	: click to center the map
• **Zoom**	: the Tool for manually zooming
• **Select**	: the Tool for selecting nodes or roads
• **Node**	: the Tool for creating nodes
• **Road**	: the Tool for creating roads
• **Drive lanes tool**	: for adjusting the number of drive lanes
• **The Help button**	: quick access to the documentation

- The Tool Specific Box appears when the node tool is selected with

  - Edge node
  - Traffic Lights
  - No Signs

After the creation of map, we can run the Validation to check for inconsistencies. Id numbers will appear next to Nodes, which will be of use later on in Statistics, or an Error dialog pointing you to errors.

One will have to correct errors manually, by following the pointer ID's to the villain node and creating extra roads. Edge nodes will have to be connected to precisely one junction, and junctions should not form dead ends with edge nodes.

Also, instead of using junctions to wrap around existing map structure sections, make turns. This is not only more consistent than artificial "junctions" for evaluation, but will display more logically.

When expanding an infrastructure it is common to have to remove. Now, with your map or one chosen out of the ready made maps, you are ready to start simulating. Start the easy traffic Simulator and load your chosen map.

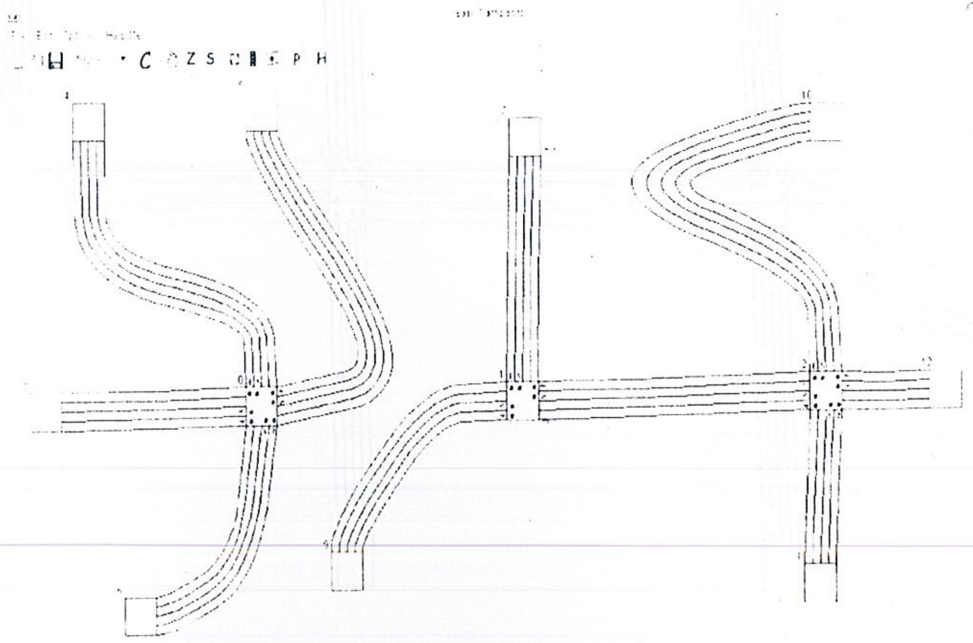
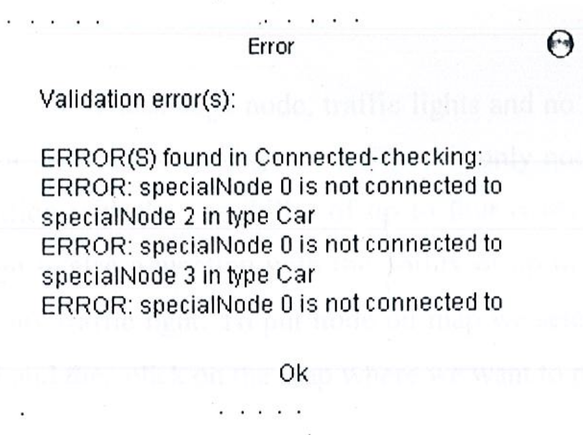


Figure 5.3: An Invalid Map (will show error when validated)


Error

Validation error(s):

ERROR(S) found in Connected-checking:
ERROR: specialNode 0 is not connected to
specialNode 2 in type Car
ERROR: specialNode 0 is not connected to
specialNode 3 in type Car
ERROR: specialNode 0 is not connected to

Ok

Figure 4.4: Validation Error

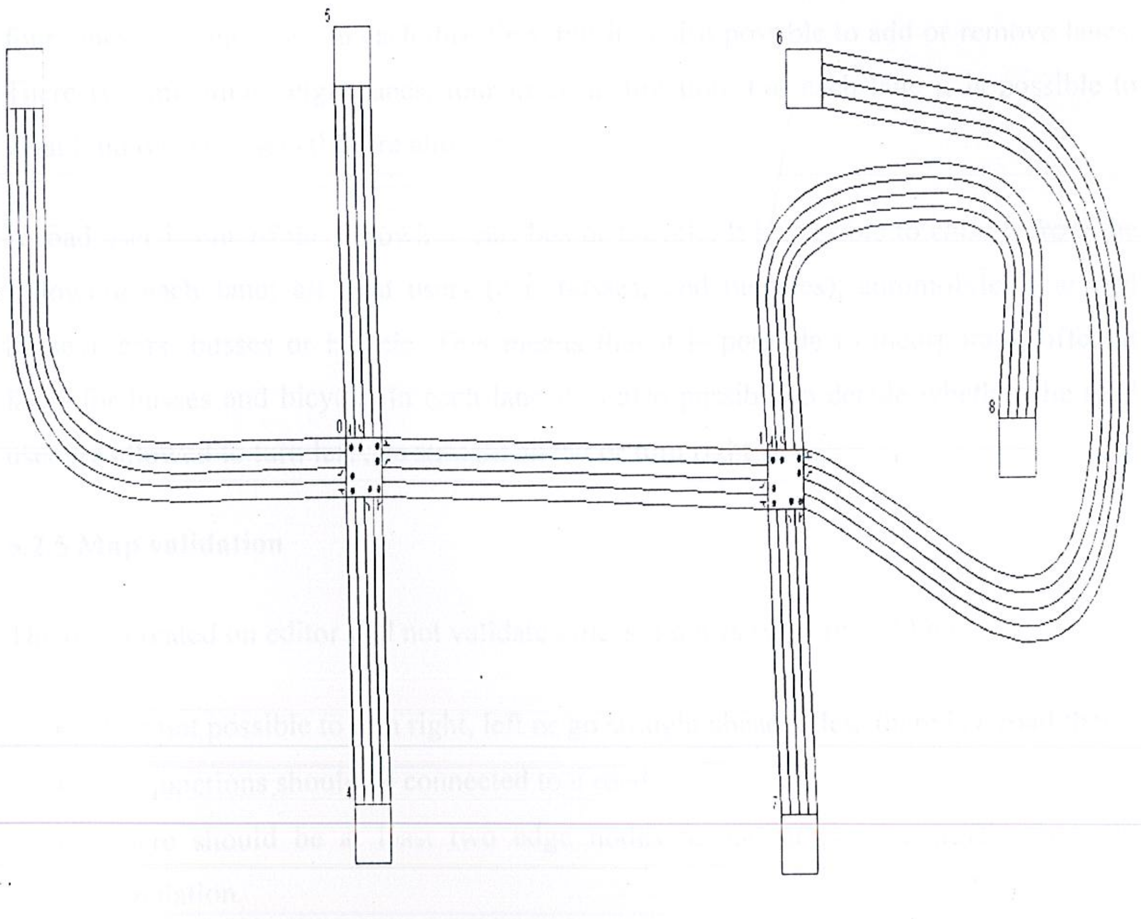


Figure 5.5: A valid Map which can be saved

5.2.3 Nodes

There are three types of nodes: edge node, traffic lights and no sign. The edge nodes are where all the traffic comes from and goes to. There is only node to these nodes. Traffic lights are the junction with the possibility of up to four roads to connect to them. The node called no sign is also a junction with the ability of up to four roads connecting to them but without any traffic light. To put node on map we select the node tool, chooses which type of tool and then click on the map where we want to place the node.

5.2.4 Roads

To put roads into the map you simply select the road tool and click in the first node and then click in the second node that you want to connect with roads. As standard there are four lanes in a road, two in each direction, but it is also possible to add or remove lanes. There is a maximum eight lanes, four in each direction. For each lane it is possible to what kind of road users that are allowed.

A road user is one of the following: car, bus or bicycle. It is possible to choose from the following each lane: all road users (car, busses, and bicycles), automobiles (car and busses), cars, busses or bicycle. This means that it is possible to incorporate different lanes for busses and bicycle. In each lane it is also possible to decide whether the road user are allowed to turn left , go straight ahead or turn right.

5.2.5 Map validation

The map created on editor will not validate unless it obeys some rules like :

- It is not possible to turn right, left or go straight ahead unless there is a road there.
- All junctions should be connected to a road.
- There should be at least two edge nodes to add or remove traffic from the simulation.
- From every edge node it should be possible to find a route to all other edge nodes.

- To should be possible for all road users to reach an edge node (else the traffic would be stuck in the map and the map would be filled up)

It is not possible to save a map that is not validated. This assures that when you open a map in the simulator, you a are guaranteed that it is possible to run the map.

5.3 The Simulator

The Simulator is where the real exploration of traffic interaction takes place. In the Simulator window the loaded map is animated with its road users, the traffic lights show their status, and various on-road signs depict the road's special driving rules.

The simulator looks a lot like a editor. We have the possibility to load or save a map. It is not possible to change the layout of the map in the simulator; this can only be done in editor.

The only changes that can be applied in the Simulator are those concerning the Driving Policy algorithms, the spawning rules for road users, and the portion of the map and zoom factor of depiction.

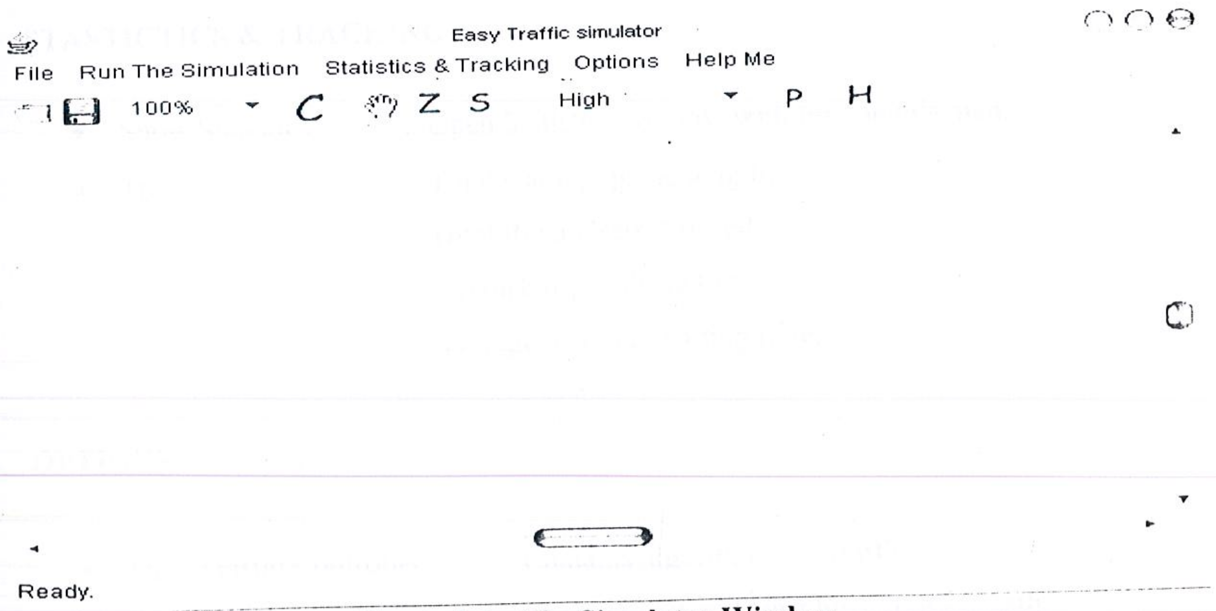


Figure 5.6 : The Simulator Window

5.3.1 The simulator Interface

The Menus in the Simulator are as follows:

FILE

• Open	: for opening a file.
• Save	: for saving a map or simulation.
• Properties	: for editing additional info on a map.
• Quit	: for quitting easy traffic.

RUN THE SIMULATION

• Run	: continuously advance the simulation.
• Pause	: stop, but not reset the simulation.
• Stop	: halt and reset the simulation.
• Speed	: toggles speed (low, high, medium, max)

STASTICTICS & TRACKING

• Show Statistics	: Open Statistics window with info on this map.
• Track	: Total waiting queue length.
	Total Road Users Arrived.
	Average trip waiting time.
	Average junction waiting time.

OPTIONS

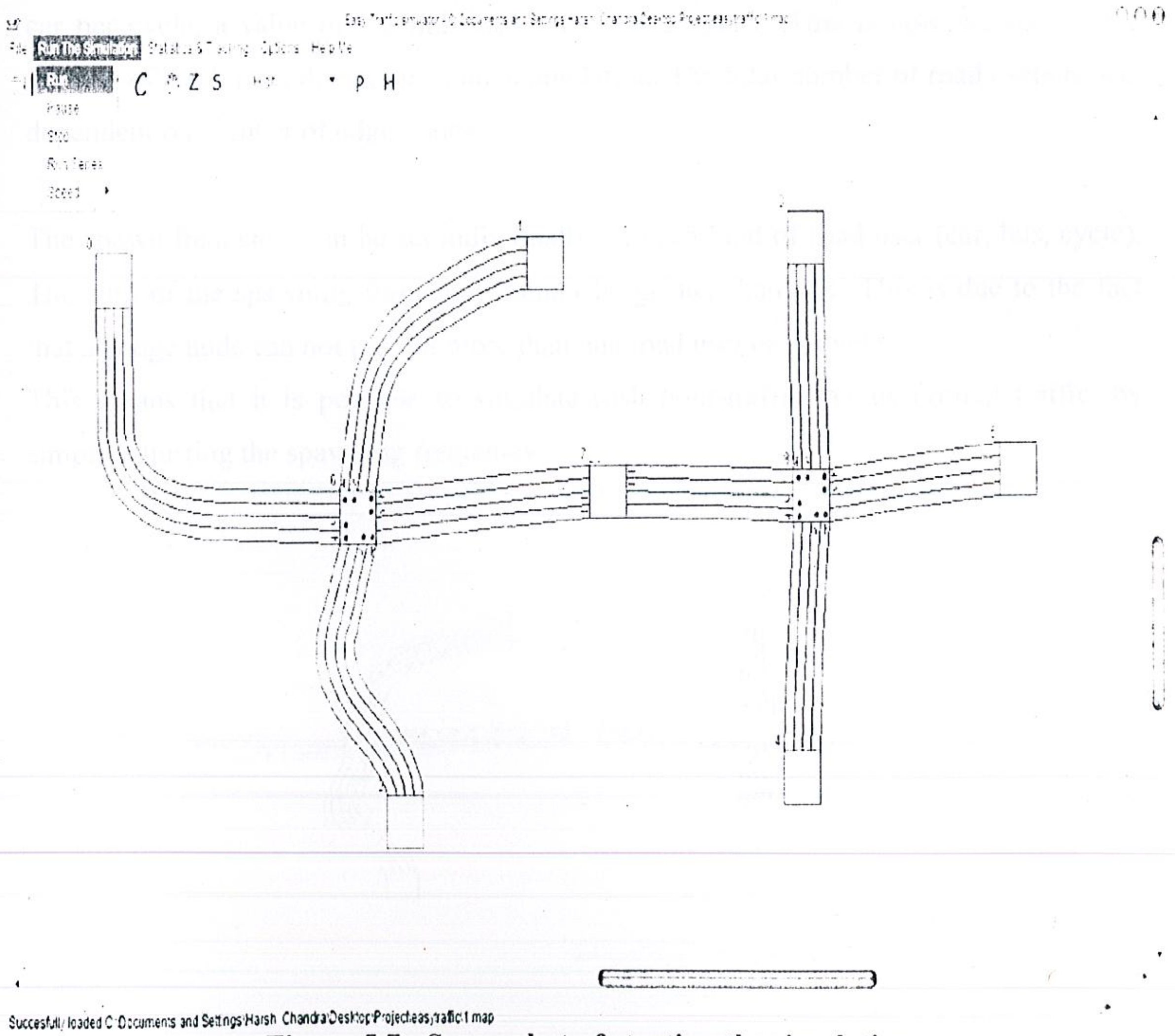
• Easy Traffic Controller	: Contains algorithms for traffic.
• Driving Policy	: shortest past and least busy shortest path.

HELP ME

• Help	: aid for the common user
• Technical Details	: precise guide to the easy traffic code
• Website	:opens the easy traffic website in a browser
• About	:a little note about easy traffic

5.3.2 Starting the simulation

To start a simulation we simple open a map that we have saved with the editor. Then we have the option to choose which *driving policy* and which *traffic controller* we want. Then we just have to click run the simulation.


Easy Traffic simulator - C:\Documents and Settings\Harsh Chandra\Desktop\Project\easytraffic\1.map
File
Run The Simulation
Statistics & Testing
Options
Help Me
Run
Pause
Stop
Run Series
Speed
C
Z
S
P
H

Succesfully loaded C:\Documents and Settings\Harsh Chandra\Desktop\Project\easytraffic\1.map

Figure 5.7 : Screenshot of starting the simulation

A simulation is divided into cycles, so it is possible to make a step wise simulation only taking one cycle at a time. This gets tedious of one wants to do some real simulation, but can be useful if one wants to observe what happens in each step. The other possibility is to run the simulation in continuous mode. Here it is possible to choose between four different simulation speed (Low, Medium, High and maximum).

5.3.3 Spawning frequency

One of the most important things in the simulator is spawning frequencies. The spawn frequency is set for each edge node. It is a value between 0 and 1 that indicates how many road users that will be initiated from that node per cycle. A value of 1 indicates 1 car per cycle, a value of 0.2 indicates 1 car per 5 cycles. This is how we control the number of road users that comes into a simulation. The total number of road users is also dependent on number of edge nodes.

The spawn frequency can be set individually for each kind of road user (car, bus, cycle). The sum of the spawning frequency cannot be greater than one. This is due to the fact that an edge node can not put out more than one road user each cycle.
This means that it is possible to simulate rush hour-traffic versus normal traffic, by simply adjusting the spawning frequency.

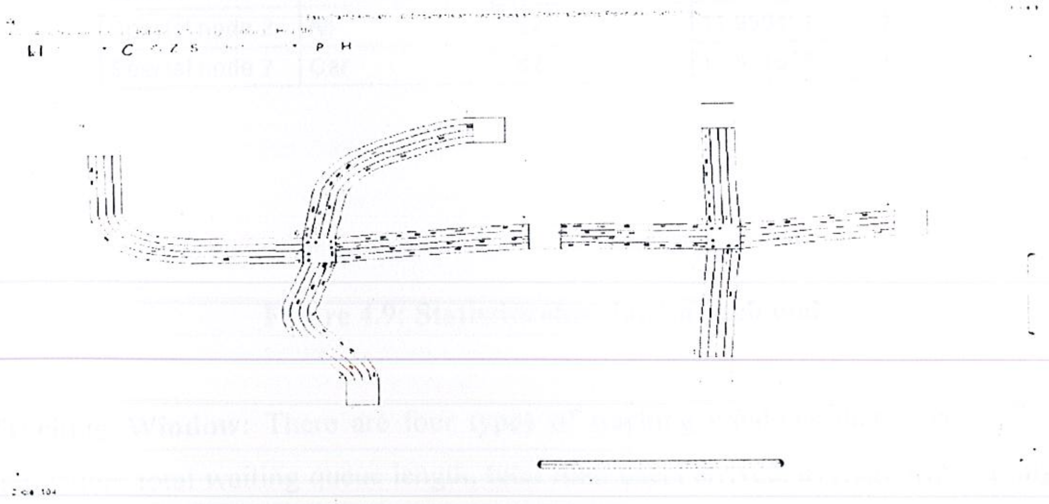
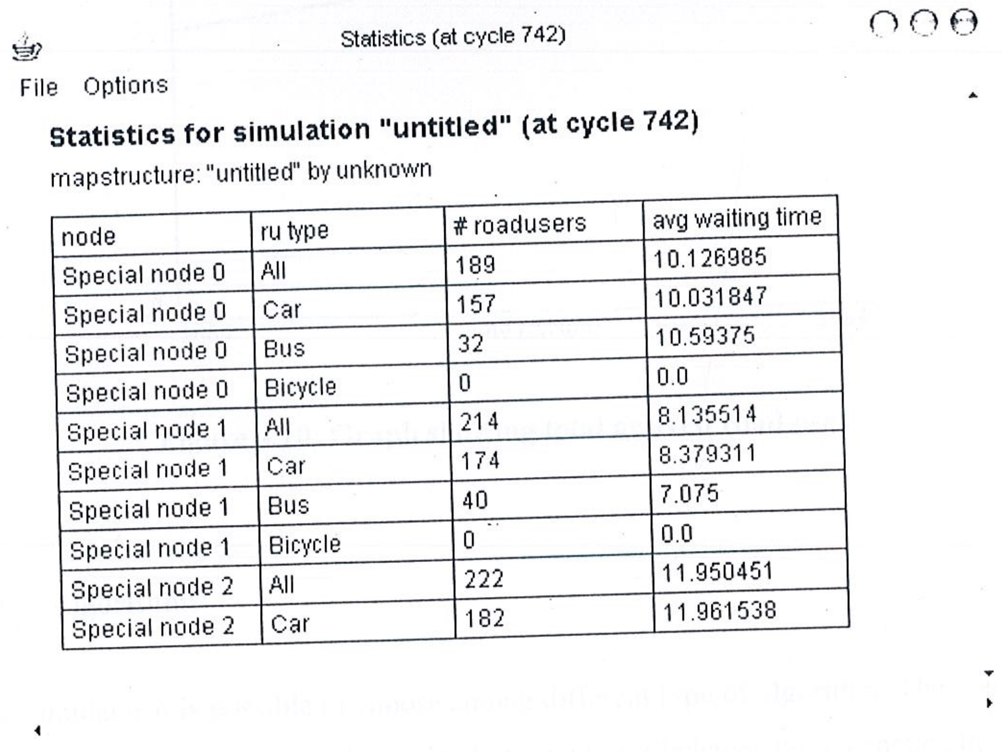


Figure 5.8: A map populated with cars having spawning frequency

5.3.4 Evaluation Tools

Another important aspect of the simulator is to gather the information about the simulation. After creating a map and running a simulation, it is time to crunch numbers: The statistics can be viewed as tables for precision, or switched to graphics for quick overview.

Statistics window: The statistics window gives you some information about the current simulation and map. It includes general information on edge nodes and junction.



Statistics for simulation "untitled" (at cycle 742)

mapstructure: "untitled" by unknown

node	ru type	# roadusers	avg waiting time
Special node 0	All	189	10.126985
Special node 0	Car	157	10.031847
Special node 0	Bus	32	10.59375
Special node 0	Bicycle	0	0.0
Special node 1	All	214	8.135514
Special node 1	Car	174	8.379311
Special node 1	Bus	40	7.075
Special node 1	Bicycle	0	0.0
Special node 2	All	222	11.950451
Special node 2	Car	182	11.961538

Figure 4.9: Statistics showing on each node.

Tracking Window: There are four types of tracking windows that covers all of the simulation: total waiting queue length, total road users arrived, average trip waiting time and average junction waiting time.

In junction it is possible to track the number of road users that have crossed it and the waiting time. In edge nodes it is possible to track the waiting queue length, number of road users arrived and the average trip waiting time.

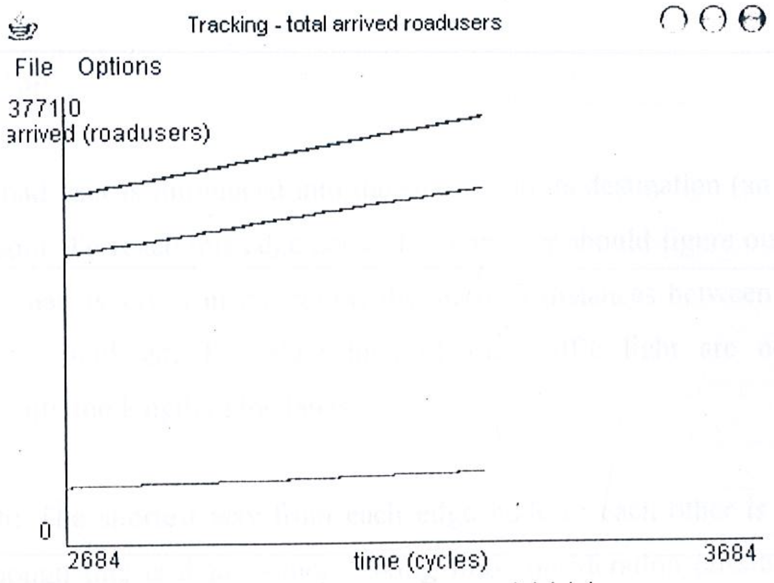


Figure 4.10: Graph showing total arrived road user

5.4 The Algorithms

In the simulator it is possible to choose among different type of algorithm. There are four different types of algorithm to determine how road user behaves three genetic algorithms for the traffic light controlling.

The variables in easy traffic consist of the road users and the way groups of traffic lights behave. Driving policies and traffic light controllers describe the algorithms used to coordinate the behavior of these. It isn't necessary to fully understand the way the variables for spawning and such affect the flow of traffic. Rather, it is worth knowing what kind of road network needs what kind of sophistication in its traffic light controller to sustain a certain density of traffic.

Eventually, implementation of physical sensors in real-world road infrastructure will be expensive, so deciding which controller to implement is a matter of balancing its effectiveness on traffic and its cost.

5.4.1 Driving Policy

When a new road user is introduced into the simulation its destination (an edge node) is chosen at random. To reach this edge node, the road user should figure out a way to get there. When a map is saved in the editor, the shortest distances between every pair of edge node are calculated. The algorithms of the traffic light are not taken into consideration, only the length of the lanes.

Shortest Path: The shortest way from each edge node to each other is calculated for every map, though this is done without taking into consideration possible traffic light control algorithms, but only the length of subsequent lanes. For every type of road user the lanes that it is able to use to get closer to its destination are computed from node to node until one full path is saved for those two edge nodes.

Road users follow the shortest path of their spawning point to their randomly assigned destination, receiving for every decision at a junction the next lane to head to. When multiple lanes qualify for best choice, one is chosen randomly.

Least Busy Shortest Path: In addition to the heuristics of Shortest path, the number of waiting road users on the next lane is taken into account when a decision has to be made between multiple lanes that have an equally short path so that the decision will be weighed according to the waiting time that (partially) blocked lanes will cause.

In short, this algorithm is that when two lanes appear to be equally good, the one with the lowest number of waiting road users taken.

Co-learning: Road users choose their next lane when crossing a junction so as to minimize probable waiting time at the next junction.

If a next lane has traffic lights, a co-learn value is requested from the traffic light control algorithm, if it is the kind to compute co-learning values. Along with this the road users sends a value for itself to the controller so other road users may benefit from it.

Co-learning will greatly improve the distribution of road users on a map with many possible shortest paths (say, a grid-like map structure), but will have little effect if there are few possibilities or the traffic load is too high to distribute much. When traffic is unpredictable though, co-learning is less useful.

Aggressive: This version of the algorithm makes it possible for the road users to switch lanes on a normal road (not just on junction)

5.4.2 Traffic Light Controllers

There are simple rules for traffic lights on one node, and complex ways of regulating a whole map structure of them. To achieve an optimum traffic flow for a specific map, it is necessary to adjust general algorithms to that map - in the case of traffic light controllers, returning an evaluation of the situation at a junction that becomes more overall effective as more road users pass.

There are number of genetic algorithms, *the ACGJ ones*, which adapt to a specific flow of traffic by preferring the road users with lowest move/wait ratio when evolving over generations.

In this section we will use the following functions and variables:

- Q - The total expected waiting time before all traffic lights for each road user, discounted for the time a light is green.

- V - The average waiting time until destination, disregarding traffic light decision.
- W - The gain of a decision of a traffic light, which sets two specific lights to green
- P - Probability that a certain other junction is reached if the current critical light is green or red
- K - number of cars waiting before a certain traffic light

Here is the short outline of implemented strategies :

- **Random:** The traffic lights are based on a random schedule. Essentially assigns a random value to each junction node every time slot, disregarding any data on road users or infrastructure. This value is a reward R for setting a traffic light on green, as seen in the change in waiting time for road users.

  As it is random here, it has little meaning, and traffic flow suffers from this on all but the simplest of maps in comparison with other algorithms. The reason it might be of use, besides its simplicity to implement (no communication) and for comparison, is its robustness. Facing large amounts of spawning road users, a random algorithm will continue to get some to their destination, albeit inefficiently.

  On a map structure as straightforward as Simple, there appears to be little difference in the flow of traffic, even with medium traffic flow. But as soon as it is used on a more complicated infrastructure over a longer period of time, one can notice how the disregarding of data about road users works out.

  The randomizations gives, over a longer time frame, each traffic light about the same period for green as for red, making busy junctions clog up and free ones stay green uselessly. This is best demonstrated by giving the edge nodes on one side high spawning frequencies and the edge nodes on another low one. Given an amount of traffic, the side with high frequencies will clog up in most cases in which the randomizations assigns it values about equal to the side with low frequencies.

This may seem a useless algorithm, but it is fairly close to what is used in current controllers!

- **ACGJ-1:** This algorithm creates for every new cycle of iteration a genetic population and tries to find the optimal city-wide configuration. This algorithm prevents deadlocks and stimulates green waves. This algorithm prevents endless waiting of road users.

- **ACGJ-2:** This algorithm, ACGJ-2, handles, when it starts, like Longest Queue, but it can do more things:

  1. It prevents infinite waiting of Road users.

  2. It tries to create green waves between busy nodes. The green wave-factor is editable as a parameter.

  3. This algorithm has a keep green factor. This editable factor is an extra weight to keep the lights green, which were green in the previous cycle.

- **ACGJ-3:** This algorithm will receive a Road user-waiting-factor per traffic light, as well as a Alternation-factor per Traffic Light. These are calculated from the genes the ACGJ3Individual has. The waiting-factor is allowed to be between 0 and 4, the alternation-factor between 0 and 2.

Per Traffic light the algorithm keeps track of the build-up Gain as calculated in previous iterations. The new gain in this so-called 'bucket' is calculated in this way:

B (t): the gain-bucket for Traffic Light't'
W (t): the number of Road users waiting at the lane of Traffic Light't'
R(t) : the weight-per-Road user as given by the genes of Traffic Light
A(t) : the alternation-factor as given by the genes of Traffic Light 't'

$$b(t) \leftarrow b(t) + \sum_{i=0}^{w(t)} r(t) \quad * \, a(t)^{i}$$

Without leaking some from the bucket, this would lead to infinite gains for each traffic light. To make sure this never happens, whenever a Road user passes a Traffic Light, the Bucket is emptied some. The following formula shows how:

$$b(t) \leftarrow b(t)[\, w(t) - 1] \, / \, w(t)]$$

$$w(t) \leftarrow w(t) - 1$$

Whenever a Road user is allowed to travel by it's Traffic Light, but can't change to it's desired Drive lane as it's full then action is taken to make it more likely that that particular lane will move and make space.

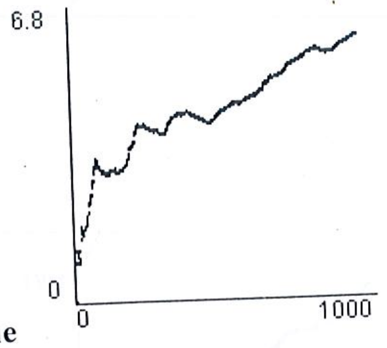
$$b(t') \leftarrow b(t') + b(t)[(\, w(t) - 1) \, / \, w(t)$$

Currently an individual represents a whole city, all the traffic lights with their independent Weights and Factors. This means that converging of the performance of this algorithm will go erratic and slowly. However, during this period many promising individuals will be created. The user is encouraged to create his own variants of this algorithm using different genetic-functions for evolving.

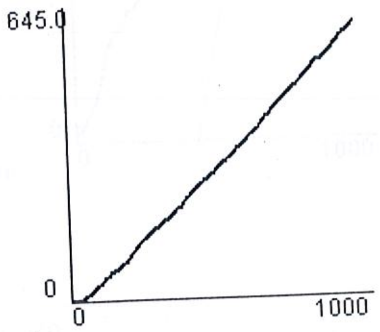
5.5 Performance Statistics

We executed a map and recorded the performance of time with respect to each genetic algorithm. Following are the graphs plotted:

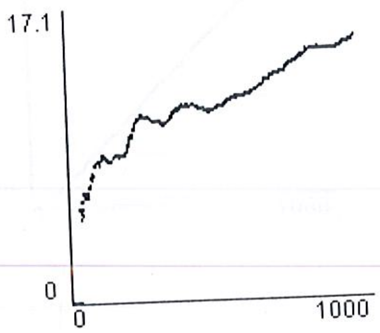
(a) ACGJ 1



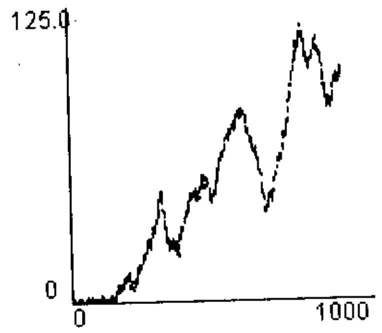
Average Junction Waiting Time



Total Road Users Arrived

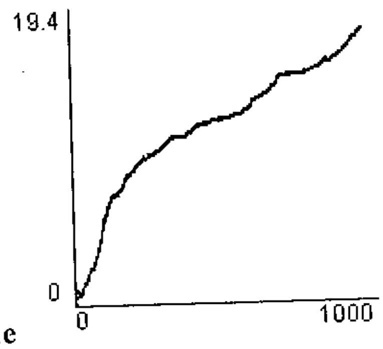


Average Trip Waiting Time

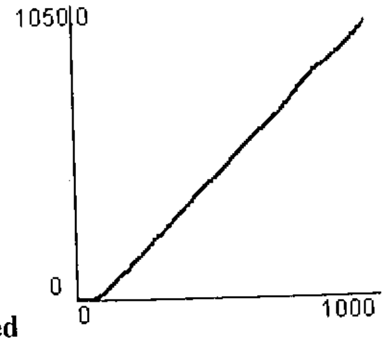


Total Waiting Queue Length

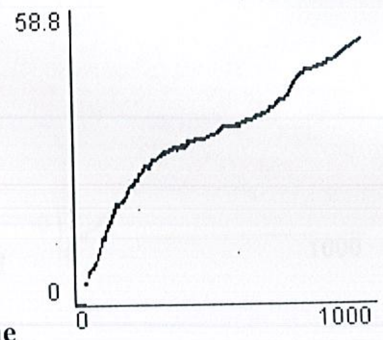
(b) ACGJ-2



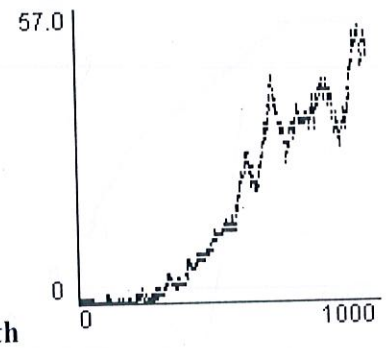
Average Junction Waiting Time



Total Road Users Arrived

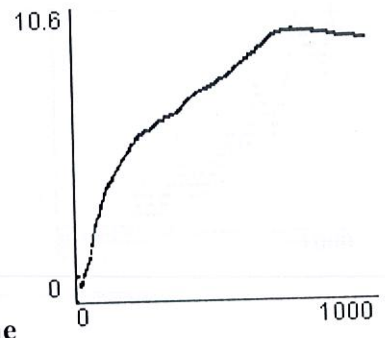


Average Trip Waiting Time

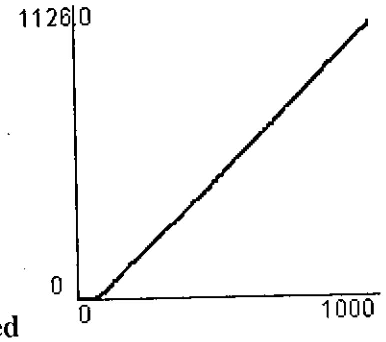


Total Waiting Queue Length

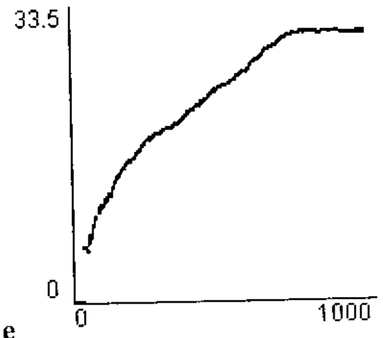
(c) ACGJ-3



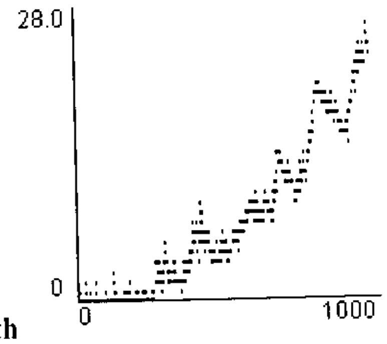
Average Junction Waiting Time



Total Road Users Arrived



Average Trip Waiting Time



Total Waiting Queue Length

Chapter 6: Limitations of Easy Traffic

Easy Traffic is only a small project, and the true power of traffic simulation and its analysis may only be unlocked in future versions. Even though it has some limitations we could not have made a simulator with so many features in the time given.

As in all larger programs there are some errors in the simulator. Especially the simulator makes errors when you connect edge nodes with no-sign nodes and let them congest the roads with a lot of traffic. The simulation can run, but will sooner or later make an error that causes the simulation to crash. We have not been able to correct this error.

There is no modeling of speed and actual defined by us in the simulator. Actually we were more interested in observing the co-operational behavior of traffic lights and the cars.

6.1 Drawbacks of Simulation

Simulation is the not the real thing or accurate, so why even try to simulate? Often when you are doing simulation you have to make a lot of assumption that do not hold in the real life. It can also be very difficult and time consuming to make a model of the real world.

Even though the simulation is not accurate it can give us a pretty good idea about what works. And the simulators have improved over years, getting us closer to closer to the real thing. It is often expensive or in some cases even impossible to try out things in real life, whereas a simulation can be quite inexpensive in comparison.

But we have to keep in mind that **simulations are not the real thing** no matter how good it is.

6.2 Expansion Possibilities

When we were creating the simulator, frankly speaking we don't know where to stop. When is there enough functionality? The problem is that one can never get too much functionality. There are lots of things that we would have liked to have implemented as well, but we did not have the time for. So here are some of the things that we think still are missing.

6.2.1 Destination Decision

Instead of making a random decision on which edge node a car goes to, this could have been implemented in many other ways.

One way could be to let edge node have a swallow frequency. This could be a number between 0 and 1, which says how many cars can go to this node. If the number is 0.5 it means that every 2 cycles the node will accept one car. The way this could be implemented is to say that instead of choosing a destination node randomly when we are creating a new road user, we instead choose a destination node with the probability relative to the swallow frequency. A problem with this approach is that you somehow should guarantee that the accumulated swallow frequency is the same as accumulated spawn frequency. Otherwise cars would get stuck in the map.

6.2.2 Trains

Another thing there is missing is the possibilities to include train in the simulation. It is not that much a problem that there exists train. But some times you have to wait for a train to pass and this indeed influence the flow of traffic.

6.2.3 Pedestrians

A thing like pedestrians is also overlooked. They might have not that big influence on the traffic flow, but once in a while a car has to stop because a pedestrian needs to cross he road.

6.2.4 Advanced Junction

Currently the traffic lights in the simulator are either red or green and road users always obey the traffic signal. To make more real, the possibility for a traffic right to turn orange should be included.

6.2.5 Advanced Roads

The car moves with the standard speed no matter where they are going. A possibility to let the road have different speed limits would be beneficial of one is trying to model traffic more precise.

Appendix A: Definitions of Terms Used In Easy Traffic

Agent: road users and traffic lights. These interact in the set infrastructure by using the data they hold on it and other agents to achieve their goal. When the data that is collected by an agent is used by other agents, of other types, their learning behavior is called a co-learning strategy.

Cycle: the distinct time measurement unit, according to these speeds (animation and tracking speed depending on algorithm and computer system):

- Low: 1 cycle/second
- Medium: 2.5cycles/second
- High: 20 cycles/second
- Maximum: 100 cycles/second

ID: each node is assigned an integer during validation to uniquely identify it. Used by validation to indicate infrastructure omissions and in statistics to point info. ID numbers start at 0; special nodes are numbered first, afterwards normal nodes.

Map structure: nodes, with traffic lights or rules, connected by roads. Validation decides if the whole fits technical rules, but the user holds ultimate responsibility. In addition to sound map-making; tweaking the spawn frequencies is necessary to gain useful info from the statistics of a simulation.

Map: a valid infrastructure containing nodes and roads properly linked. Further usefulness may be defined by the mapmaker having given it proper traffic lights or rules, and balancing the type and number of road sides and edge nodes. Only maps can be loaded is the simulator because they are known to be valid.

Node: a junction between 1-4 roads.

- **Edge Node:** has only one road and is the only node type with spawns road users, and is also the only node type that can be tweaked with in the simulator, by changing the spawn frequency.

- **Junction:** links roads and provides rules or traffic lights for control.
- **Spawning Frequency:** value between 0 and 1 indicating how many road users will be initiated from the edge node the value belongs to per cycle. A value of 1 indicates one per cycle, 0.5 indicates 1 per 2 cycles, and so on.
- **Traffic Light:** each lane link to a junction node must have one, or have rules.
- **No Signs:** node without traffic lights - has only rules. Useful for forcing certain traffic flows.

Road: must always link two nodes. Road sides and borders are separated b_{ij} a black line, and border between Lanes b_{ij} a grey line. Turns have only a visual representation and no influence on the behavior of road users; because these have no acceleration (they don't slow down in turns).

Road User: symbolizes a car, truck, or bus. Road users start (spawn) at an edge node and follow a certain shortest path to reach their destination, observing the rules at a road and traffic lights. At a junction, the traffic light setting, shortest path, eventual rules and the availability of road space decides whether it moves on. A road user is not intelligent, its path is.

- **Automobile:** subset of road users with a motorized vehicle - not allowed on the lanes reserved for cyclists.
- **Bus:** has a greater size than normal cars, and may gain priority at a junction because it carries more passengers.
- **Car:** Have a size of 2 and a normal speed of 2.
- **Cyclist:** Have a size of 1 and a normal speed of 1. .
- **Passenger:** used in some algorithms to give priority to road users that transports more.

- **Truck:** same as bus, only it gains no priority for extra passengers.

Traffic Light: represented by one dot at the end of a lane's direction in a junction, which can turn red or green, green meaning proceed if your destination road has free space, red meaning stay. Road users always follow the rules of a light.

Appendix B:
Source Code of Genetic Algorithm applied in Easy Traffic

1. ACGJ 1

```
package tsim.algo.etc;

import tsim.*;
import tsim.sim.*;
import tsim.algo.etc.*;
import tsim.map.*;
import tsim.utils.*;
import tsim.xml.*;

import java.io.IOException;
import java.util.*;
import java.awt.Point;

/*
This algorithm creates for every new cycle of iteration a genetic population and tries to
find the optimal city-wide configuration. This algorithm prevents deadlocks and
stimulates green waves. This algorithm prevents endless waiting of roadusers. The fitness
function "calcfitness" is the most important function of this algorithm.

*/
```

```
public class genetic1 extends etcontroller implements XMLSerializable, TwoStageLoader,
InstantiationAssistant
{
        Population genPopulation;
        int ruMoves;
        int avgWaitingTime;
        protected mapstructure map;
        public final static String shortXMLName="etc-acgj1";
        protected InstantiationAssistant assistant;
        protected static float mutationFactor = 0.05f;
        protected static int populationSize   = 200;
        protected static int maxGeneration    = 100;

        public genetic1(mapstructure i)
        {       super(i);
        }
        public void setmapstructure(mapstructure i)
        {       super.setmapstructure(i);
                genPopulation = new Population(i);
                genPopulation.initialize();
                ruMoves = 0;
        }


        /* Calculates how every traffic light should be switched  parameter :  The
  TLDecision is a tuple consisting of a traffic light and a reward (Q) value, for it to be
  green

         */
        public TLDecision[][] decideTLs()
        {
            int maxLength, num_lanes, num_nodes, maxId, temp_len, ru_pos;
            num_nodes = tld.length;

            for (int i=0; i < num_nodes; i++)
            {
                    maxLength = -1;
                    maxId = -1;
                    num_lanes = tld[i].length;
                    for(int j=0; j < num_lanes; j++)
tld[i][j].setGain(tld[i][j].getTL().getLane().getNumRoadusersWaiting());
            }

            try
            {
                    genPopulation.resetGeneration();
                    genPopulation.evaluateGeneration();
                    for (int i=0; i<maxGeneration; i++)
                    {
                            if (genPopulation.createNextGeneration()) break;
                            genPopulation.evaluateGeneration();
                    }
            }
            catch (Exception e)
            {
                    System.out.println(e+"");
                    e.printStackTrace();
            }

            genPopulation.sortMembers();
            Person p = genPopulation.getFirstPerson();
            if (p!=null)
```

```
                {
                        try{p.fillTld(tld);}
                        catch(Exception e)
                        {System.out.println(e);e.printStackTrace();}
                }

                ruMoves=0;
                return tld;
        }

        public void updateRoaduserMove(Roaduser _ru, Drivelane _prevlane, Sign
_prevsign, int _prevpos, Drivelane _dlanenow, Sign _signnow, int _posnow, PosMov[]
posMovs, Drivelane desired)
        {
            ruMoves++;
        }

        private class Population implements XMLSerializable, TwoStageLoader
        {       Vector members;
                mapstructure map;
                Random rnd;
                float currentMax;
                int numMaxTimes;

                public Population(mapstructure map){
                        this.map = map;
                        members = new Vector();
                        rnd = new Random();
                        currentMax = 0;
                        numMaxTimes = 0;
                        initialize();
}

// Initializes this population
public void initialize() {
        members.removeAllElements();
        if (ACGJ1.populationSize<10) ACGJ1.populationSize = 10;
        for (int i=0; i<ACGJ1.populationSize; i++) {
                Person p = new Person(map, rnd);
                p.randomizeData();
                members.addElement(p);
        }
        currentMax=0;
        numMaxTimes = 0;
}

public void resetGeneration()
{       float total = members.size();
        float current = 0;
        for (Enumeration e=members.elements(); e.hasMoreElements();) {
                Person p = (Person) e.nextElement();
                if (rnd.nextFloat()<(current/total)) p.randomizeData();
                current++;
        }
        currentMax = 0;
        numMaxTimes = 0;
}

public void evaluateGeneration() throws mapException
{       calcFitnesses();        }

public boolean createNextGeneration()
```

```
{       int popSize = members.size();
        if (calcRelativeFitnesses()) return true;

// Kill some members of this population, about one half is killed here
            for (Iterator i = members.iterator(); i.hasNext();)      {
                    Person p = (Person) i.next();
                    if
(p.relFitness<4*rnd.nextFloat()*rnd.nextFloat()*rnd.nextFloat())
                            i.remove();
            }

            // Generate new childs
            sortMembers();
            int memSize = members.size();
            if (memSize==0) initialize();
            else {
                    while (members.size()<popSize) {
                            float rand = rnd.nextFloat();
                            int p1 = (int) ((1.0f-rand*rand)*memSize);
                            rand = rnd.nextFloat();
                            int p2 = (int) ((1.0f-rand*rand)*memSize);
                            if (p1>=memSize || p2 >=memSize) continue;
                            Person parent1 = (Person) members.elementAt(p1);
                            Person parent2 = (Person) members.elementAt(p2);
                            Person child = generateChild(parent1,parent2);
                            members.addElement(child);

                    }

            }

            // Mutate this generation
            for (Enumeration e=members.elements(); e.hasMoreElements();) {
                Person p = (Person) e.nextElement();
                mutatePerson(p);
        }

        return false;
}

private void calcFitnesses() throws mapException
{       for (Enumeration e=members.elements(); e.hasMoreElements();) {
                Person p = (Person) e.nextElement();
                p.calcFitness();
        }
}

private boolean calcRelativeFitnesses()
{       float min = 0;
        float max = 0;

        boolean first = true;
        for (Enumeration e=members.elements(); e.hasMoreElements();) {
                Person p = (Person) e.nextElement();
                if (first || p.fitness<min) {
                        min = p.fitness;
                }
                if (first || p.fitness>max) {
                        max = p.fitness;
                }
                first = false;
        }
        if (min==max) return true;
        if (max-min<0.01) return true;
        for (Enumeration e=members.elements(); e.hasMoreElements();) {
                Person p = (Person) e.nextElement();
                p.relFitness = (p.fitness-min)/(max-min);
        }

        if (max==currentMax) {
                numMaxTimes++;
                if (numMaxTimes>4) return true;
        }
        else {
                numMaxTimes=0;
                currentMax=max;
        }

        return false;
}

public Person generateChild(Person parent1, Person parent2)
{
        Person child = new Person(map, rnd);
        for (int i=0; i<child.ndinf.length; i++)
        {
                // choose random one parent
                int config = parent1.ndinf[i].config;
                if (rnd.nextFloat()>=0.5) config = parent2.ndinf[i].config;
                child.ndinf[i].config=config;
        }
        return child;
 }
```

```
public void mutatePerson(Person person)
{
        if (rnd.nextFloat()<=mutationFactor)
        {
                int mutNode = (int) (rnd.nextFloat()*person.ndinf.length);
                mutateNodeInfo(person.ndinf[mutNode]);

                // Another mutation is possible too
                if (mutationFactor<0.8) mutatePerson(person);
        }
}

public void mutateNodeInfo(NodeInfo ndi)
{
        ndi.config = (int) (rnd.nextFloat()*ndi.configsize);
}

public void sortMembers()
{
        //sort members so that the first has the highest fitness

        Person p1 =null, p2 =null;

        for (int i=0; i<members.size();i++)
        {
                p1 = (Person) members.elementAt(i);
                for (int j=members.size()-1; j>=i; j--)
                {
                        p2 = (Person) members.elementAt(j);
                        // if p2>p1...
                        if (p2.fitness>p1.fitness)
                        {
                                members.setElementAt(p2,i);
                                members.setElementAt(p1,j);
                                p1=p2;
                        }
                }
        }

    }

    public Person getFirstPerson()
    {
            if (members.size()==0) return null;
            return (Person)(members.elementAt(0));
    }

    // XMLSerializable implementation of Population

    public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
    {       numMaxTimes=myElement.getAttribute("max-
times").getIntValue();
            currentMax=myElement.getAttribute("current-
max").getFloatValue();
            members=(Vector)(XMLArray.loadArray(this,loader,assistant));
    }

    public XMLElement saveSelf () throws XMLCannotSaveException
    {       XMLElement result=new XMLElement("population");
            result.addAttribute(new XMLAttribute("current-
max",currentMax));
```

```
                        result.addAttribute(new XMLAttribute("max-
times",numMaxTimes));
                        return result;
                }

                public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
                {       XMLArray.saveArray(members,this,saver,"members");
                }

                public String getXMLName ()
                {       return "model.etc.population";
                }

        public void setParentName (String parentName_) throws XMLTreeException
                {       throw new XMLTreeException
                                ("Operation not supported. ACGJ1Population has a fixed
parentname");
                }
public void loadSecondStage (Dictionary dictionaries) throws
XMLTreeException,XMLInvalidInputException
                {       XMLUtils.loadSecondStage(members.elements(),dictionaries);
                }
        }
                private class Person implements XMLSerializable, TwoStageLoader
        {
                mapstructure map;
                NodeInfo [] ndinf;
                Random rnd;
                float fitness;
                float relFitness;
        protected String myParentName="model.etc.population";

        public Person(mapstructure map, Random rnd)
        {
                this.map = map;
                this.rnd  = rnd;
                Node [] allNodes = map.getAllNodes();
                ndinf = new NodeInfo[allNodes.length];
                for (int j=0; j<allNodes.length; j++)
                {
                        Node nd = allNodes[j];
                        ndinf[nd.getId()]=new NodeInfo(nd);
                }

                fitness = -1;
                relFitness = -1;
        }

        public void randomizeData()
        {
                for (int i=0; i<ndinf.length; i++)
                {
                        NodeInfo ndi = ndinf[i];
                        ndi.config  = (int)(ndi.configsize *rnd.nextFloat());
                }
        }
/** This is the fitness-function, it now returns the number of cars that can drive with the
given config.*/
public void calcFitness() throws mapException
{
```

```
float totalFitness = 0.0f;

for (int i=0; i<ndinf.length; i++)
{               NodeInfo ndi = ndinf[i];
                Junction ju = null;
               if (ndi.nd instanceof Junction) ju = (Junction) ndi.nd; else continue;
                Sign [] config = ju.getSignConfigs()[ndi.config];
                for  (int j=0; j<config.length; j++)
{       int numRUWaiting=config[j].getLane().getNumRoadusersWaiting();
        if (numRUWaiting>0)
{       Roaduser ru = config[j].getLane().getFirstRoaduser();
        totalFitness +=1.0 + 0.1*ru.getDelay();  // prevents infinite waiting times

}
        totalFitness += numRUWaiting * 0.3f;

}
// Stimulate green waves, if a next lane is also green, give an extra reward
for (int l=0; l<config.length; l++)
{
        Drivelane dl = config[l].getLane();
        Drivelane [] dls = dl.getSign().getNode().getLanesLeadingFrom(dl,0);
                        for (int j=0; j<dls.length; j++)
                        {
                                Sign s2      = dls[j].getSign();
                               NodeInfo ndi2 = ndinf[s2.getNode().getId()];
                                if (!(ndi2.nd instanceof Junction)) continue;
                                Sign [] cfg2  = ((Junction)
(ndi2.nd)).getSignConfigs()[ndi2.config];
                                for (int k=0; k<cfg2.length; k++)
                                {
                                        if (cfg2[k]==s2) totalFitness+=0.1;
                                }
                }
            }
        }

        fitness = totalFitness;
    }

    public void fillTld(TLDecision [][] tld) throws mapException
    {
        for (int i=0; i<ndinf.length; i++)
        {
            NodeInfo ndi = ndinf[i];
            int nodeID = ndi.nd.getId();
            setConfiguration(ndi,tld[nodeID]);
        }
    }

    private void setConfiguration(NodeInfo ndi, TLDecision [] tl) throws
mapException
    {
        if (tl.length<=0) return;
        Junction ju = null;
        if (ndi.nd instanceof Junction) ju = (Junction) ndi.nd; else return;

        Sign [] config = ju.getSignConfigs()[ndi.config];
        for (int j=0; j<tl.length; j++)
        {
            tl[j].setGain(0);
            for (int k=0; k<config.length; k++)
                if (tl[j].getTL()==config[k]) tl[j].setGain(1);
            }
    }

    // XMLSerializable implementation of Person

    public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
    {       fitness=myElement.getAttribute("fitness").getFloatValue();
            relFitness=myElement.getAttribute("rel-fitness").getFloatValue();
            ndinf=(NodeInfo[])XMLArray.loadArray(this,loader,assistant);
    }

    public XMLElement saveSelf () throws XMLCannotSaveException
    {       XMLElement result=new XMLElement("person");
            result.addAttribute(new XMLAttribute("fitness",fitness));
            result.addAttribute(new XMLAttribute("rel-fitness",relFitness));
            return result;
    }

    public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
    {       XMLArray.saveArray(ndinf,this,saver,"node-info");
    }

    public String getXMLName ()
    {       return myParentName+".person";
    }

    public void setParentName (String newParentName)
    {       myParentName=newParentName;
    }
```

```
                // TwoStageLoader implementation of Person

                public void loadSecondStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
                {       XMLUtils.loadSecondStage(new
ArrayEnumeration(ndinf),dictionaries);
                }



        }

        private class NodeInfo implements XMLSerializable, TwoStageLoader
        {
                Node nd;
                int config;
                int configsize;

                protected String myParentName="model.etc.population.person";
                protected TwoStageLoaderData loadData=new TwoStageLoaderData();

                // Empty constructor for loading
                public NodeInfo ()
                {}

                public NodeInfo(Node nd)
                {       this.nd = nd;
                        config  = -1;
                        if (nd instanceof Junction) configsize = ((Junction)
 nd).getSignConfigs().length; else configsize=0;
                }
        // XMLSerializable implementation of NodeInfo

        public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
        {       config=myElement.getAttribute("config").getIntValue();
                configsize=myElement.getAttribute("config-size").getIntValue();
                loadData.nodeId=myElement.getAttribute("node-
id").getIntValue();
        }

        public XMLElement saveSelf () throws XMLCannotSaveException
        {       XMLElement result=new XMLElement("nodeinfo");
                result.addAttribute(new XMLAttribute("config",config));
                result.addAttribute(new XMLAttribute("config-size",configsize));
                result.addAttribute(new XMLAttribute("node-id",nd.getId()));
                return result;
        }

        public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
        {       // NodeInfo objects don't have child objects
        }

        public String getXMLName ()
        {       return myParentName+".nodeinfo";
        }

        public void setParentName (String newParentName)
        {       myParentName=newParentName;
        }
```

```
            // TwoStageLoader implementation of NodeInfo

            class TwoStageLoaderData
            {       int nodeId;
            }

            public void loadSecondStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
            {       nd=(Node)((Dictionary)dictionaries.get("node")).get(new Integer
                            (loadData.nodeId));
            }


    }

    public void showSettings(Controller c)
    {
            String[] descs = {"Population Size", "Maximal generation number",
"Mutation factor"};
            float[] floats = {mutationFactor};
            int[] ints = {populationSize, maxGeneration};
            etcSettings settings = new etcSettings(descs, ints, floats);

            settings = doSettingsDialog(c, settings);

            mutationFactor = settings.floats[0];
            populationSize = settings.ints[0];
            maxGeneration  = settings.ints[1];

    }
```

```
        // XMLSerializable implementation

        public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
        {       super.load(myElement,loader);
                ruMoves=myElement.getAttribute("ru-moves").getIntValue();
                avgWaitingTime=myElement.getAttribute("avg-waittime").getIntValue();
                mutationFactor=myElement.getAttribute("mut-factor").getFloatValue();
                populationSize=myElement.getAttribute("pop-size").getIntValue();
                maxGeneration=myElement.getAttribute("max-gen").getIntValue();
                genPopulation=new Population(map);
                loader.load(this,genPopulation);
        }

        public XMLElement saveSelf () throws XMLCannotSaveException
        {       XMLElement result=super.saveSelf();
                result.setName(shortXMLName);
                result.addAttribute(new XMLAttribute("ru-moves",ruMoves));
               result.addAttribute(new  XMLAttribute("avg-waittime",avgWaitingTime));
                result.addAttribute(new XMLAttribute("mut-factor",mutationFactor));

                result.addAttribute(new         XMLAttribute("pop-size",populationSize));
                result.addAttribute(new XMLAttribute("max-gen",maxGeneration));
                return result;
        }

        public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
        {       super.saveChilds(saver);
                saver.saveObject(genPopulation);
        }
        public String getXMLName ()
        {       return "model."+shortXMLName;
        }

        // TwoStageLoader implementation
        public void loadSecondStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
        {       super.loadSecondStage(dictionaries);
                genPopulation.loadSecondStage(dictionaries);
        }

        // InstantiationAssistant implementation

        public Object createInstance (Class request) throws
            ClassNotFoundException,InstantiationException,IllegalAccessException
        {       if (Population.class.equals(request))
                { return new Population(map);
                }
                else if (Person.class.equals(request))
                { return new Person(map,genPopulation == null ? new Random () :
genPopulation.rnd);
                }
                else if (NodeInfo.class.equals(request))
                { return new NodeInfo();
                }
                else
                { throw new ClassNotFoundException
                  ("ACGJ1 InstantiationAssistant cannot make instances of "+ request);
                }
        }
        public boolean canCreateInstance (Class request)
        {       return Population.class.equals(request) || Person.class.equals(request) ||
NodeInfo.class.equals(request);
        }
}
```

2. ACGJ-2

```
package tsim.algo.etc;
import tsim.*;
import tsim.sim.*;
import tsim.algo.etc.*;
import tsim.map.*;
import tsim.utils.*;
import tsim.xml.*;
import java.io.IOException;
import java.util.Random;
import java.util.*;
import java.awt.Point;

/** This algorithm will, when it starts, handle like Longest Queue, but it can do more
things:
 * 1. It prevents infinite waiting of Roadusers
 * 2. It tries to create green waves between busy nodes. The green_wave-factor is editable
as a parameter.
 * 3. The algorithm can be stimulated to develop patterns in TL-configuration-settings,
therefore it remembers the last 100 configurations. This pattern factor is also editable
 * 4. This algorithm has a keep green factor. This editable factor is an extra weight to
keep the lights green, that were green in the previous cycle.
 */
```

```
public class ACGJ2 extends etcontroller implements
XMLSerializable,TwoStageLoader,InstantiationAssistant
{
        protected static final String shortXMLName="etc-acgj2";
        protected NodeInfo [] ndinf;

        protected static float PAT_FACTOR=0;
        protected static float KEEP_GREEN_FACTOR = 0.05f;
        protected static float GREEN_WAVE_FACTOR = 2.0f;
        protected static float LOOK_AHEAD_FACTOR = 1.0f;
        protected NextCycles bestSoFar;

        public ACGJ2(mapstructure i)
        {       super(i);
                bestSoFar = null;

                for (int j=0; j<ndinf.length; j++)
                {
                        NodeInfo ndi = ndinf[j];
                        for (int k=0; k< tld.length; k++)
                        {
                                TLDecision [] tl = tld[k];
                                if (tl.length>0)
                                        if (tl[0].getTL().getNode()==ndi.nd) ndi.tldIndex =
k;

                                for (int l=0; l<ndi.dli.length; l++)
                                {       DrivelaneInfo dl = ndi.dli[l];
                                        for (int m=0; m<tld[k].length; m++)
                                        {       if (tld[k][m].getTL()==dl.dl.getSign())
dl.tldIndex=m;
```

```
                                    }
                              }
                        }
                  }


      }

      public void setmapstructure(mapstructure i)
      {     super.setmapstructure(i);
            Node [] allNodes = i.getAllNodes();
            ndinf = new NodeInfo[allNodes.length];
            for (int j=0; j<ndinf.length; j++) ndinf[j]=new NodeInfo(allNodes[j], this);
      }

      /* Calculates how every traffic light should be switched
       * parameter The TLDecision is a tuple consisting of a traffic light and a reward
(Q) value, for it to be green
       *
       */
      public TLDecision[][] decideTLs()
      {
            for (int i=0; i<ndinf.length; i++) ndinf[i].updateVariables();

            NodeInfo [] sortedNdi = sortNodeInfo(ndinf);

            // Normal weights
            for (int i=0; i<sortedNdi.length; i++)
            {
                  NodeInfo currentNode = sortedNdi[i];
                  if (currentNode.tldIndex!=-1)
```

```
                {
                        currentNode.calcPatValues(PAT_FACTOR);
                        for (int j=0; j<currentNode.dli.length; j++)
                        {
                                DrivelaneInfo currentDL = currentNode.dli[j];
                                if (currentDL.tldIndex!=-1)
                                {
                                        float qval = currentNode.getQValue(j);
                                        qval += currentNode.pat[j];
                                        if (currentDL.dl.getSign().mayDrive()) qval
*= (1+KEEP_GREEN_FACTOR);


        tld[currentNode.tldIndex][currentDL.tldIndex].setGain(qval);

                                }
                                else System.out.println("Negative index");
                        }
                }
        }

        return tld;
    }
    public NodeInfo [] sortNodeInfo(NodeInfo [] ndi)
    {
        NodeInfo result[] = new NodeInfo[ndi.length];
        for (int i=0; i<result.length; i++) result[i]=ndi[i];
        // bubble sort algorithm
        for (int j=result.length; j>0; j--)
```

```
                {
                        for (int i=0; i<j-1; i++)
                        {
                                if (result[i].currentBusyness>result[i+1].currentBusyness)
                                {
                                        //swap
                                        NodeInfo temp=result[i];
                                        result[i]=result[i+1];
                                        result[i+1]=temp;
                                }
                        }
                }

                return result;
        }
        public void updateRoaduserMove(Roaduser _ru, Drivelane _prevlane, Sign
_prevsign, int _prevpos, Drivelane _dlanenow, Sign _signnow, int _posnow, PosMov[]
posMovs, Drivelane desired)
        {
                // No implementation necessary
        }

        private class NextCycles implements XMLSerializable
        {       int cyclesForward;
                mapstructure currentmap;
                protected String myParentName="model.etc";

                public NextCycles ()
                {       currentmap=map;
                }
```

```
            // XMLSerializable implementation of NextCycles

            public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
            {       cyclesForward=myElement.getAttribute("cycles-
forward").getIntValue();
            }

            public XMLElement saveSelf () throws XMLCannotSaveException
            {       XMLElement result=new XMLElement("nextcycles");
                    result.addAttribute(new XMLAttribute("cycles-
forward",cyclesForward));
                    return result;
            }

            public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
            {       // A NextCycles object has no child objects
            }

            public String getXMLName ()
            {       return myParentName+".nextcycles";
            }

            public void setParentName (String newParentName)
            {       myParentName=newParentName;
            }
       }

       private class NodeInfo implements XMLSerializable, TwoStageLoader
       {       float currentWaiting;
```

```
float currentBusyness;
Configuration [] lastConfigs; //history of the last 10 configs
DrivelaneInfo [] dli, dlo;  //i=inbound o=outbound
Node nd;
int tldIndex;
Sign [] bestSignSoFar;
float [] pat;
ACGJ2 acgj;

// XML stuff
String myParentName="model.etc";
TwoStageLoaderData loadData=new TwoStageLoaderData();

// Empty constructor for loading
public NodeInfo (ACGJ2 acgj)
{        this.acgj=acgj;
}

public NodeInfo(Node nd, ACGJ2 acgj)
{
        this.acgj=acgj;
        tldIndex = -1;
        currentWaiting = 0;
        currentBusyness=0;

        this.nd = nd;
        lastConfigs = new Configuration[10];
        for (int i=0; i<lastConfigs.length; i++) lastConfigs[i]=null;

        // create dli
        Drivelane [] dls;
```

```
                try
                {
                        dls = nd.getInboundLanes();
                }
                catch(Exception e)
                {
                        dls = new Drivelane[0];
                }
                dli = new DrivelaneInfo[dls.length];
                for (int i=0; i<dli.length; i++)
                {
                        dli[i]=new DrivelaneInfo(dls[i],this,acgj);
                }

        try
        {
                dls = nd.getInboundLanes();
        }
        catch(Exception e)
        {
                dls = new Drivelane[0];
        }
        dlo = new DrivelaneInfo[dls.length];
        for (int i=0; i<dlo.length; i++)
        {
                dlo[i]=new DrivelaneInfo(dls[i],this, acgj);
        }

                bestSignSoFar = null;
                pat = new float[dli.length];
        }
```

```
public int getId()
{       return nd==null ? loadData.nodeId : nd.getId();
}

public void updateVariables()
{
        // update waiting and busyness
        currentWaiting = 0;
        Drivelane [] dls;
        try
        {
                dls = nd.getInboundLanes();
        }
        catch(Exception e)
        {
                dls = new Drivelane[0];
        }
        for (int i=0; i<dls.length; i++)
        {
                currentWaiting += dls[i].getNumRoadusersWaiting();
        }
        currentBusyness = currentBusyness * 0.9f + currentWaiting * 0.1f;

        // update lastConfigs
        boolean [] green = new boolean[dls.length];
        int numGreen = 0;
        for (int i=0; i< green.length; i++)
        {
                if (dls[i].getSign().mayDrive())
                {
```

```
                numGreen++;
                green[i]=true;
            }
            else green[i]=false;
        }

        boolean [] currentGreenSigns = new boolean[green.length];
        int j=0;
        for (int i=0; i<green.length; i++)
        {
            if (green[i])
            {
                currentGreenSigns[j] = dls[i].getSign().mayDrive();
                j++;
            }
        }

        if (lastConfigs[0]!=null)
        if (lastConfigs[0].signEquals(currentGreenSigns))
        {
            lastConfigs[0].incNumCycles();
        }
        else
        {
            // Shift all configs
            for (int i=lastConfigs.length-2; i>=0; i--)
                lastConfigs[i+1]=lastConfigs[i];
        }
    lastConfigs[0]=new Configuration(currentGreenSigns, 1);
        bestSignSoFar = null;
    }
```

```
        public float getQValue(int DLindex)
        {
                DrivelaneInfo dlInf= dli[DLindex];
                int result = dlInf.dl.getNumRoadusersWaiting();
                float laFactor =
dlInf.getLookAheadDisc()*acgj.LOOK_AHEAD_FACTOR;

                return result + laFactor;
        }

        /** Calculates the pattern values*/
        public void calcPatValues(float patFactor)
        {
                for (int i=0; i<pat.length; i++) pat[i]=0;
                if (lastConfigs[0]==null) return;
                if (patFactor==0) return;

                boolean [] currentSigns = lastConfigs[0].signs;
                Configuration [] temp = new Configuration[lastConfigs.length];

                int count=0, totCycles=0;
                for (int i=1; i<temp.length; i++)
                {
                        if (lastConfigs[i]!=null)
                        {
                                if (lastConfigs[0].signEquals(lastConfigs[i].signs))
                                {
                                        temp[i]=lastConfigs[i];
                                        count++;
                                        totCycles+=lastConfigs[i].numCycles;
```

```
                    }
                    else
                    temp[i]=null;
                }
                else break;
            }

            if (totCycles>0)
            for (int i=1; i<temp.length; i++)
            {
            if (lastConfigs[i]!=null)
            {
                if (lastConfigs[0].signEquals(lastConfigs[i].signs))
                {
                    int leftSecs = 0;
                    Configuration old=null;
                    if
(lastConfigs[i].numCycles>lastConfigs[0].numCycles)
                    {
                        leftSecs = lastConfigs[i].numCycles-
lastConfigs[0].numCycles;
                        old = lastConfigs[i];
                    }
                    else
                    {
                        old = lastConfigs[i-1];
                        leftSecs = lastConfigs[i].numCycles;
                    }
                    boolean [] sgs=old.signs;
                    for (int j=0; j<dli.length; j++)
                    {
```

```
                                        DrivelaneInfo dl = dli[j];
                                        for (int k=0; k<sgs.length; k++)
                                        {
                                                if
(dl.dl.getSign().mayDrive()==sgs[k])
                                                {

      pat[dl.tldIndex]+=patFactor*(leftSecs/totCycles);
                                                }
                                        }
                                }
                        }
                        else
                        temp[i]=null;
                }
                else break;
                }


        }

        // XMLSerializable implementation of NodeInfo

        public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
        {
     lastConfigs=(Configuration[])XMLArray.loadArray(this,loader,acgj);
                dli=(DrivelaneInfo[])XMLArray.loadArray(this,loader,acgj);
                dlo=(DrivelaneInfo[])XMLArray.loadArray(this,loader,acgj);
                if ( ! myElement.getAttribute("signs-null").getBoolValue())

     bestSignSoFar=(Sign[])XMLArray.loadArray(this,loader,acgj);
```

```
                        pat=(float[])XMLArray.loadArray(this,loader);
                        currentWaiting=myElement.getAttribute("current-
waiting").getFloatValue();
                        currentBusyness=myElement.getAttribute("current-
busy").getFloatValue();
                        tldIndex=myElement.getAttribute("tld-index").getIntValue();
                        loadData.nodeId=myElement.getAttribute("node-
id").getIntValue();
                        // Set NodeInfo in DrivelaneInfo
                        Enumeration di1=new ArrayEnumeration(dli);
                        while (di1.hasMoreElements())
                        {       ((DrivelaneInfo)(di1.nextElement())).setNodeInfo(this);
                        }
                        Enumeration di2=new ArrayEnumeration(dlo);
                        while (di2.hasMoreElements())
                        {       ((DrivelaneInfo)(di2.nextElement())).setNodeInfo(this);
                        }
                }

                public XMLElement saveSelf () throws XMLCannotSaveException
                {       XMLElement result=new XMLElement("nodeinfo");
                        result.addAttribute(new         XMLAttribute("current-
waiting",currentWaiting));
                        result.addAttribute(new         XMLAttribute("current-
busy",currentBusyness));
                        result.addAttribute(new         XMLAttribute("tld-index",tldIndex));
                        result.addAttribute(new XMLAttribute("node-id",nd.getId()));
                        result.addAttribute(new         XMLAttribute("signs-
null",bestSignSoFar==null));
                        return result;
                }
```

```
            public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
            {       XMLArray.saveArray(lastConfigs,this,saver,"last-configs");
                    XMLArray.saveArray(dli,this,saver,"dl-inbound");
                    XMLArray.saveArray(dlo,this,saver,"dl-outbound");
                    if (bestSignSoFar != null)
                    {       XMLUtils.setParentName(new
    ArrayEnumeration(bestSignSoFar),getXMLName());
                            XMLArray.saveArray(bestSignSoFar,this,saver,"best-
sign");
                    }
                    XMLArray.saveArray(pat,this,saver,"pat");
            }

            public String getXMLName ()
            {       return myParentName+".nodeinfo";
            }

            public void setParentName (String newParentName)
            {       myParentName=newParentName;
            }

            // TwoStageLoader implementation of NodeInfo

            class TwoStageLoaderData
            {       int nodeId;
            }

            public void loadSecondStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
```

```
				{	nd=(Node)((Dictionary)dictionaries.get("node")).get(new Integer
						(loadData.nodeId));
					if (bestSignSoFar != null )
						XMLUtils.loadSecondStage(new
ArrayEnumeration(bestSignSoFar),dictionaries);
				}

				public void loadThirdStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
				{	XMLUtils.loadSecondStage(new
ArrayEnumeration(dli),dictionaries);
					XMLUtils.loadSecondStage(new
ArrayEnumeration(dlo),dictionaries);
				}

			}

			private class DrivelaneInfo implements XMLSerializable,TwoStageLoader
			{
				Drivelane dl;
				NodeInfo ndi; // node with sign at this DL
				private NodeInfo otherNode; // node without sign at this DL
				int [] target;  // 0 = left 1= straight 2=right
				float [] drivers;
				int tldIndex;
				ACGJ2 acgj;

				// XML stuff
				String myParentName="model.etc.nodeinfo";
				TwoStageLoaderData loadData=new TwoStageLoaderData();
```

```
// Empty constructor for loading
public DrivelaneInfo (ACGJ2 acgj)
{       this.acgj=acgj;
}

public DrivelaneInfo(Drivelane dl, NodeInfo ndi, ACGJ2 acgj)
{
        this.dl=dl;
        this.ndi=ndi;
        this.otherNode = null;
        this.acgj=acgj;
        tldIndex = -1;

        boolean [] targets= dl.getTargets();
        int count=0;
        for (int i=0; i<targets.length; i++) if (targets[i]) count++;
        target = new int[count];
        int count2=0;
        for (int j=0; j<targets.length; j++) if (targets[j])
        {
                target[count2]=j;
                count2++;
        }

        drivers=new float[count];
        for (int i=0; i<drivers.length; i++) drivers[i]=0;
}

public void setNodeInfo (NodeInfo ni)
{       ndi=ni;
}
```

```
public NodeInfo getOtherNode()
{
        if (otherNode!=null) return otherNode;
        Node an = dl.getRoad().getAlphaNode();
        Node bn = dl.getRoad().getBetaNode();
        Node cn = null;
        if (ndi.nd==an) cn=bn; else cn=an;
        for (int i=0; i<acgj.ndinf.length; i++)
        {
                if (acgj.ndinf[i].nd==cn)
                {
                        otherNode = acgj.ndinf[i];
                        return otherNode;
                }
        }
        return null;
}

public float getTurnChance(int direction)
{
        return 1.0f/target.length;
}

public float getLookAheadDisc()
{
ListIterator li = dl.getQueue().listIterator();
Roaduser ru = null;
int pos = 0;
int ru_pos;
int count = 0;
while (li.hasNext())
```

```
        {
                ru = (Roaduser) li.next();
                ru_pos = ru.getPosition();
                if (ru_pos > pos) break;
                else if (ru_pos == pos) {
                        pos += ru.getLength();
                        count++;
                }
        }

        float laFactor=0.0f;

        while (li.hasNext())
        {
                ru = (Roaduser) li.next();
                ru_pos = ru.getPosition();
                laFactor -= (1/(ru_pos-pos));
        }
        return laFactor;
        }

        // XMLSerializable implementation of DrivelaneInfo

        public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
        {       tldIndex=myElement.getAttribute("tld-index").getIntValue();
                loadData.nodeId=myElement.getAttribute("other-
node").getIntValue();
                loadData.laneId=myElement.getAttribute("lane-id").getIntValue();
                target=(int[])XMLArray.loadArray(this,loader);
                drivers=(float[])XMLArray.loadArray(this,loader);
```

```
}

public XMLElement saveSelf () throws XMLCannotSaveException
{       XMLElement result=new XMLElement("laneinfo");
        result.addAttribute(new XMLAttribute("tld-index",tldIndex));
        result.addAttribute(new XMLAttribute("other-node" ,
        otherNode == null ? -1 : otherNode.getId()));
        result.addAttribute(new XMLAttribute("lane-id",
                        dl == null ? -1 :  dl.getId()));
        return result;
}

public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
{       XMLArray.saveArray(target,this,saver,"target");
        XMLArray.saveArray(drivers,this,saver,"drivers");
}

public String getXMLName ()
{       return myParentName+".laneinfo";
}

public void setParentName (String newParentName)
{       myParentName=newParentName;
}

// TwoStageLoader implementation of DrivelaneInfo
class TwoStageLoaderData
{       int nodeId,laneId;
}
public void loadSecondStage (Dictionary dictionaries)
```

```
            {       otherNode=(NodeInfo)((Dictionary)dictionaries.get("node-
info")).get(new Integer
                            (loadData.nodeId));
dl=(Drivelane)((Dictionary)dictionaries.get("lane")).get(new Integer(loadData.laneId));
            }

        }
        private class Configuration implements XMLSerializable
        {
            protected boolean [] signs;  // red=false, green=true
            protected int numCycles;
            protected String myParentName="model.etc.nodeinfo";

            // Empty constructor for loading
            public Configuration () {}

            public Configuration (Sign [] s, int numCycles)
            {
                //this.signs     = signs;
                signs = new boolean[s.length];
                for (int i=0;i<s.length;i++)
                    signs[i]=s[i].mayDrive();
                this.numCycles = numCycles;
            }

            public Configuration (boolean [] b, int numCycles)
            {
                //this.signs     = signs;
            this.signs = b;
            this.numCycles = numCycles;
            }
```

```
public void incNumCycles()
{
        numCycles++;
}

public boolean signEquals(boolean [] sgs)
{
        if (sgs.length!=signs.length) return false;
        int count=0;
        for (int i=0; i< signs.length; i++)
        {
                for (int j=i; j< sgs.length; j++)
                {
                        if (signs[i]==sgs[j]) count++;
                }
        }
        if (count!=signs.length) return false;
        return true;
}

// XMLSerializable implementation of NextCycles

public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
{       numCycles=myElement.getAttribute("num-cycles").getIntValue();
        signs=(boolean[])XMLArray.loadArray(this,loader);
}

public XMLElement saveSelf () throws XMLCannotSaveException
```

```
            {       XMLElement result=new XMLElement("config");
                    result.addAttribute(new  XMLAttribute("num-cycles",numCycles));
                    return result;
            }


            public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
            {       XMLArray.saveArray(signs,this,saver,"signs");
            }


            public String getXMLName ()
            {       return myParentName+".config";
            }


            public void setParentName (String newParentName)
            {       myParentName=newParentName;
            }


        }
        public void showSettings(Controller c)
        {
            String[] descs = {"Pattern factor", "Keep green light factor", "Green wave
factor", "Look ahead factor"};
            float[] floats = {PAT_FACTOR,
KEEP_GREEN_FACTOR,GREEN_WAVE_FACTOR,LOOK_AHEAD_FACTOR};
            etcSettings settings = new etcSettings(descs, null, floats);

            settings = doSettingsDialog(c, settings);
            PAT_FACTOR = settings.floats[0];
            KEEP_GREEN_FACTOR = settings.floats[1];
            GREEN_WAVE_FACTOR = settings.floats[2];
```

```
                LOOK_AHEAD_FACTOR = settings.floats[3];
        }

        // XMLSerializable implementation

        public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
        {       PAT_FACTOR=myElement.getAttribute("pat-factor").getFloatValue();
                KEEP_GREEN_FACTOR=myElement.getAttribute("keep-
green").getFloatValue();
                GREEN_WAVE_FACTOR=myElement.getAttribute("green-
wave").getFloatValue();
                LOOK_AHEAD_FACTOR=myElement.getAttribute("look-
ahead").getFloatValue();
                ndinf=(NodeInfo[])XMLArray.loadArray(this,loader,this);
                if (! myElement.getAttribute("best-null").getBoolValue())
                {       bestSoFar=new NextCycles();
                        loader.load(this,bestSoFar);
                }
        }

        public XMLElement saveSelf () throws XMLCannotSaveException
        {       XMLElement result=super.saveSelf();
                result.setName(shortXMLName);
                result.addAttribute(new XMLAttribute("pat-factor",PAT_FACTOR));
                result.addAttribute(new XMLAttribute("keep-
green",KEEP_GREEN_FACTOR));
                result.addAttribute(new XMLAttribute("green-
wave",GREEN_WAVE_FACTOR));
                result.addAttribute(new XMLAttribute("look-
ahead",LOOK_AHEAD_FACTOR));
```

```
        result.addAttribute(new XMLAttribute("best-null",bestSoFar==null));
        return result;
    }

    public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
    {   XMLArray.saveArray(ndinf,this,saver,"node-info");
        if (bestSoFar != null )
            saver.saveObject(bestSoFar);
    }

    public String getXMLName ()
    {   return "model."+shortXMLName;
    }

    // TwoStageLoader implementation
    public void loadSecondStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
    {   super.loadSecondStage(dictionaries);
        XMLUtils.loadSecondStage(new ArrayEnumeration(ndinf),dictionaries);
        loadThirdStage(dictionaries);
    }

    // Yeah. Baby. A three-stage loader
    public void loadThirdStage (Dictionary dictionaries) throws
XMLInvalidInputException,XMLTreeException
    {   Enumeration ni=new ArrayEnumeration(ndinf);
        Dictionary nodeInfo=new Hashtable();
        NodeInfo tmp;
        while (ni.hasMoreElements())
        {   tmp=(NodeInfo)ni.nextElement();
```

```
                nodeInfo.put(new Integer(tmp.getId()),tmp);
        }
        dictionaries.put("node-info",nodeInfo);
        ni=new ArrayEnumeration(ndinf);
        while (ni.hasMoreElements())
                ((NodeInfo)(ni.nextElement())).loadThirdStage(dictionaries);
}

// InstantiationAssistant implementation

public Object createInstance (Class request) throws
    ClassNotFoundException,InstantiationException,IllegalAccessException
{       if (NextCycles.class.equals(request))
        {  return new NextCycles();
        }
        else if (NodeInfo.class.equals(request))
        {  return new NodeInfo(this);
        }
        else if (DrivelaneInfo.class.equals(request))
        {  return new DrivelaneInfo(this);
        }
        else if (Configuration.class.equals(request))
        {  return new Configuration();
        }
        else
        { throw new ClassNotFoundException
         ("ACGJ2 InstantiationAssistant cannot make instances of "+
          request);
        }
}
```

```
        public boolean canCreateInstance (Class request)
        {       return NextCycles.class.equals(request) ||
                 NodeInfo.class.equals(request) ||
                 DrivelaneInfo.class.equals(request) ||
                 Configuration.class.equals(request) ;
        }

}
```

3. ACGJ-3

```
package tsim.algo.etc;

import tsim.*;
import tsim.sim.*;
import tsim.algo.etc.*;
import tsim.map.*;
import tsim.utils.*;
import tsim.xml.*;
import java.io.IOException;
import java.util.Random;
import java.util.*;
import java.awt.Point;

public class genetic3 extends etcontroller implements
InstantiationAssistant,XMLSerializable
{
        /** A constant for the number of steps an individual may run */
        protected static int NUM_STEPS = 400;
        /** A constant for the chance that genes cross over */
        protected static float CROSSOVER_CHANCE = 0.1f;
        /** A constant for the chance that mutation occurs */
        protected static float MUTATION_CHANCE = 0.001f;
```

```
/** A constant for the number of individuals */
protected static int NUMBER_INDIVIDUALS = 50;
/** Used constants for array addressing */
protected static int BUCK = 3,
                WAIT = 2,
                DECF = 1,
                RUWF = 0;

/** counter for the number of steps the simulation has ran yet */
protected int num_step;
/** counter for how many roadusers had to wait a turn */
protected int num_wait;
/** counter for the number of roadusers that did move */
protected int num_move;
/** counter for the number of roadusers that did move */
protected int num_nodes;
/** the current ACGJ3Individual that is running  */
protected ACGJ3Individual ind;
/** The Population of ACGJ3Individuals */
protected ACGJ3Population pop;
/** The pseudo-random number generator for generating the chances that certain
events will occur */
protected Random random;

// Stuff for our XML parser
protected final static String shortXMLName="etc-acgj3";
protected InstantiationAssistant assistant=this;
```

```
/**
 * Creates a new ACGJ3 Algorithm.
 * This etc-algorithm is using genetic techniques to find an optimum in
calculating the gains for each trafficlight.  Till date it hasnt functioned that well. Rather
poorly, to just say it.
 * parameter :  i The mapstructure this algorithm will have to operate on
 */
public genetic3(mapstructure i)
{       random = new Random();
        setmapstructure(i);
}

/* Changes the mapstructure this algorithm is working on
 * parameter :  i The new mapstructure for which the algorithm has to be set up
 */
public void setmapstructure(mapstructure _i)
{       super.setmapstructure(_i);
        map = _i;
        num_nodes = tld.length;
        pop     =     new     ACGJ3Population(_i,     NUMBER_INDIVIDUALS,
100,CROSSOVER_CHANCE,MUTATION_CHANCE);
        ind = pop.getIndividual();
        num_wait = 0;
        num_step = 0;
}

/**
 * Calculates how every traffic light should be switched
 * return :             Returns a double array of TLDecision's. These are tuples of
a TrafficLight and the gain-value of when it's set to green.
 */
```

```
public TLDecision[][] decideTLs()
{
        if(num_step==NUM_STEPS) {
                pop.getNextIndividual(num_wait,num_move);
                System.out.println("Next Individual gotten, previous:(wait,move)
("+num_wait+","+num_move+")");
                num_wait = 0;
                num_step = 0;
                num_move = 0;
        }
        for(int i=0;i<num_nodes;i++) {
                int num_tl = tld[i].length;
                for(int j=0;j<num_tl;j++) {
                        float q = ind.getQValue(i,j);
                if(trackNode!=-1)
                        if(i==trackNode) {
                                Drivelane currentlane = tld[i][j].getTL().getLane();
                                boolean[] targets = currentlane.getTargets();
                                System.out.println("node: "+i+" light: "+j+" gain:
"+q+"            "+targets[0]+"            "+targets[1]+"            "+targets[2]+"
"+currentlane.getNumRoadusersWaiting());
                        }
                        if(q==Float.NaN) {
                                System.out.println("NaN!");
                        }
                        else if(q==Float.POSITIVE_INFINITY) {
                                System.out.println("Too big. I think.");
                        }
                        else if(q>=100000)
                                System.out.println("Too big.");
```

```
                    tld[i][j].setGain(q);
                }
            }

            num_step++;
            return tld;
        }

        /** Resets the algorithm */
        public void reset() {
            ind.reset();
        }

        /**
         * Provides the etc-algorithm with information about a roaduser that has had it's
go in the moveAllRoaduser-cycle.
         * From this information it can be distilled whether a roaduser has moved or had
to wait.
         *
         */
        public void updateRoaduserMove(Roaduser _ru, Drivelane _prevlane, Sign
_prevsign, int _prevpos, Drivelane _dlanenow, Sign _signnow, int _posnow, PosMov[]
posMovs, Drivelane _desired)
        {
            if(_prevsign == _signnow && _prevpos == _posnow) {
                // Previous sign is the same as the current one
                // Previous position is the same as the previous one
                // So, by definition we had to wait this turn. bad.

                num_wait += _ru.getSpeed();
```

```
                        if(_prevsign.getType()==Sign.TRAFFICLIGHT &&
_prevsign.mayDrive() == true &&
                         _desired != null &&
_desired.getSign().getType()==Sign.TRAFFICLIGHT &&
                         _desired.getNodeLeadsTo().getType()==Node.JUNCTION)
                        {
                              ind.siphonToOtherBucket((TrafficLight) _signnow,
(TrafficLight) _desired.getSign());
                        }
                  }
                  else if(_prevsign != _signnow && _signnow !=null && _prevsign
instanceof TrafficLight) {
                        //clearly passed a trafficlight.
                        //lets empty the bucket value a bit.


      ind.relaxBucket(_prevsign.getNode().getId(),(TrafficLight)_prevsign);
                        num_move += _ru.getSpeed();
                  }
                  else {
                        // Roaduser did move
                        if(_prevsign != _signnow) {
                              // Passed a Sign, thus moved max.

                              num_move += _ru.getSpeed();
                        }
                        else {
                              // Didnt pass a Sign, so might've moved lessThanMax
                              int old_move = num_move;
                              num_move += (_prevpos - _posnow);
```

```
                    num_wait += _ru.getSpeed() - (_prevpos - _posnow);

                }
            }
        }

        protected class ACGJ3Population implements XMLSerializable
        {

            /** the chance that mutation occurs when evolving. Set via Constructor of
    ACGJ3Population */
            protected float mutate;
            /*the chance that crossover occurs at evolving.  */
            protected float crossover;
            /* the current number of individuals in this population */
            protected int num_ind;
            /* the total amount of generations this population should evolve over */
            protected int num_gen;
            /** the current Individual that's showing off it's coolness */
            protected int this_ind;
            /* a counter for the current generation of Individuals */
            protected int this_gen;
            protected ACGJ3Individual[] inds;
            /* Creates a new population of ACGJ3Individuals*
             * Parameter :  map The mapstructure the population will run on
             *        _num_ind the number of individuals in the population
             *        _num_gen the number of generations there should be evolved over
             *        cross the chance that crossover occurs at reproduction
             *        mut the chance that mutation occurs at reproduction
             */
```

```
protected ACGJ3Population(mapstructure map, int _num_ind, int
_num_gen, float cross, float mut)
{
        mutate = mut;
        crossover = cross;
        num_ind = _num_ind;
        num_gen = _num_gen;
        inds = new ACGJ3Individual[_num_ind];
        for(int i=0;i<_num_ind;i++) {
                inds[i] = new ACGJ3Individual(map);
        }
        this_ind = 0;
        this_gen = 0;
}

protected ACGJ3Population()
{       // Empty constructor for loading
}

/* return the number of the current generation of individuals */
protected int getCurrentGenerationNum() {
        return this_gen;
}

/*return the current individual */
protected ACGJ3Individual getIndividual() {
        return inds[this_ind];
}

/* parameter : perf The performance of the current Individual
 * return   : the next ACGJ3Individual. Either one of this generation,
 * or if all individuals in this generation have been used,
 * it evolves into a new generation and returns a new Individual from that
generation. */
protected ACGJ3Individual getNextIndividual(int wait, int move) {
        inds[this_ind].setWait(wait);
        inds[this_ind].setMove(move);

        this_ind++;
        if(this_ind>=num_ind) {
                evolve();
                this_ind = 0;
        }
        return inds[this_ind];
}

/* BubbleSorts an array of ACGJ3Individuals on the parameter of
performance    * parameter : ar the array to be sorted
 * return    : the sorted array
 */
protected ACGJ3Individual[] sortIndsArr(ACGJ3Individual[] ar)
{
        int num_ar = ar.length;
        for(int j=0;j<num_ar-1;j++) {
                for(int i=0;i<num_ar-1-j;i++) {
                        if(ar[i].getFitness()<ar[i+1].getFitness()) {
                                ACGJ3Individual temp = ar[i+1];
                                ar[i+1] = ar[i];
                                ar[i] = temp;
                        }

                }
                }
                return ar;
        }

        /* Calculates the fitness of this Individual
         * parameter :  ind the individual of which the fitness has to be calculated
         * return    :  the fitness of this individual
         */
        protected float calcFitness(ACGJ3Individual ind) {
                float w = (float) ind.getWait();
                float g = (float) ind.getMove();
                return g/(w+g);
        }

        /* Mates two ACGJ3Individuals creating two new ACGJ3Individuals
          * parameter : ma The mummy -ACGJ3Individual
         *               pa The pappa-ACGJ3Individual
         * return    : an array of length 2 of two newborn ACGJ3Individuals
         */
    protected ACGJ3Individual[] mate(ACGJ3Individual ma, ACGJ3Individual pa)
    {
    System.out.println("Mating ("+ma.getWait()+","+ma.getMove()+") and
("+pa.getWait()+","+pa.getMove()+")");

                byte[] ma1 = ma.getReproGenes(crossover,mutate);
                byte[] pa1 = pa.getReproGenes(crossover,mutate);
                byte[] ma2 = ma.getReproGenes(crossover,mutate);
                byte[] pa2 = pa.getReproGenes(crossover,mutate);

ACGJ3Individual[] kids = {new ACGJ3Individual(ma1,pa1),new
ACGJ3Individual(ma2,pa2)};
```

```
        return kids;
    }
```

/* Evolves the current generation of ACGJ3Individuals in the ACGJ3Population into a new one. */

```
    protected void evolve()
    {
        if(this_gen<num_gen) {
            // Create a new generation.
            float[] fitnesses = new float[num_ind];
            for(int i=0;i<num_ind;i++) {
                inds[i].setFitness(calcFitness(inds[i]));
            }

            float totfit=0;
            float avgfit=Float.MIN_VALUE;
            float maxfit=-1;
            float totwait=0;
            float avgwait=0;
            float totmove=0;
            float avgmove=0;

            for(int i=0;i<num_ind;i++) {
                float fit = inds[i].getFitness();
                totfit+=fit;
                if(fit>maxfit)
                    maxfit = fit;
                int wait = inds[i].getWait();
                int move = inds[i].getMove();
                totwait+=wait;
                totmove+=move;
```

```
                    }
                    avgwait = totwait/num_ind;
                    avgmove = totmove/num_ind;
                    avgfit = totfit/num_ind;

System.out.println("Previous        gen        stats        ("+avgwait+","+avgmove+")"+"
(afit,mxfit):("+avgfit+","+maxfit+")");
                    System.out.println("Evolving...");

                    inds = sortIndsArr(inds);

                    ACGJ3Individual i1 = inds[0];
                    ACGJ3Individual i2 = inds[1];
                    ACGJ3Individual i3 = inds[2];

                    int normal_mating = (int) Math.floor(num_ind*0.8);
                    int succession_ma = (int) Math.ceil(num_ind*0.2);
                    if(normal_mating%2!=0) {
                        normal_mating--;
                        succession_ma++;
                    }

                ACGJ3Individual[] newInds = new ACGJ3Individual[num_ind];

                    for(int i=0;i<normal_mating;i+=2) {
                        ACGJ3Individual[] kids = mate(inds[i],inds[i+1]);
                        newInds[i] = kids[0];
                        newInds[i+1] = kids[1];
                    }

                    for(int i=normal_mating;i<num_ind;i++) {
                        newInds[i] = inds[i-normal_mating];
                    }

                    inds = newInds;

                    this_gen++;
                }
                else {
                    System.out.println("Done with evolving");
                    // set the best individual as the ruling champ?
                    // do nothing
                    return;
                }
            }

            // XMLSerializable implementation of ACGJ3Population

public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
            {       mutate=myElement.getAttribute("mutate").getFloatValue();
                    crossover=myElement.getAttribute("crossover").getFloatValue();
                    num_ind=myElement.getAttribute("num-ind").getIntValue();
                    num_gen=myElement.getAttribute("num-gen").getIntValue();
                    this_ind=myElement.getAttribute("this-ind").getIntValue();
                    this_gen=myElement.getAttribute("this-gen").getIntValue();

        inds=(ACGJ3Individual[])XMLArray.loadArray(this,loader,assistant);
            }

            public XMLElement saveSelf () throws XMLCannotSaveException
            {       XMLElement result=new XMLElement("population");
```

```
                        result.addAttribute(new XMLAttribute("mutate",mutate));
                        result.addAttribute(new XMLAttribute("crossover",crossover));
                        result.addAttribute(new XMLAttribute("num-ind",num_ind));
                        result.addAttribute(new XMLAttribute("num-gen",num_gen));
                        result.addAttribute(new XMLAttribute("this-ind",this_ind));
                        result.addAttribute(new XMLAttribute("this-gen",this_gen));
                        return result;
                }

public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
                {       XMLArray.saveArray(inds,this,saver,"individuals");
                }

                public String getXMLName ()
                {       return "model.etc.population";
                }

        public void setParentName (String parentName_) throws XMLTreeException
                {       throw new XMLTreeException
                        ("Operation not supported. ACGJ3Population has a fixed
parentname");
                }
        }

        protected class ACGJ3Individual implements XMLSerializable
        {
                /* The Chromosomes as gotten from mama   */
                protected byte[] bytema;
                protected byte[] bytepa;
```

```
/* The array of float values that describe this ACGJ3Individual. Hence the name of the
array.
 * Encoded are for every TrafficLight in the mapstructure:
 * a float value 'weight per waiting roaduser'
 * a float value 'degrading/increasement weight per Roaduser waiting further up the
Queue'
 * a float value 'gain-bucket' in which all the weights of waiting Roadusers are collected
 * a float value 'number of Roadusers waiting' which is needed to make decreasement of
the gain-bucket possible*/

                protected float[][][] me;
                protected int wait;
                protected int move;
                protected float fitness;

                //XML Parser stuff
                protected String myParentName="model.etc.population";

/** Creates a new ACGJ3Individual, providing the mapstructure it should run on */
                protected ACGJ3Individual(mapstructure map)
                {
                        bytema = new byte[2*map.getNumNodes()];
                        bytepa = new byte[2*map.getNumNodes()];
                        random.nextBytes(bytema);
                        random.nextBytes(bytepa);
                        createMe();
                        wait = 0;
                        move = 0;
                }

                protected void createMe() {
```

```
                me = new float[map.getNumNodes()][][];
                for(int i=0;i<map.getNumNodes();i++) {
                        me[i] = new float[tld[i].length][];
                        for(int j=0;j<tld[i].length;j++) {
        //Qweight per waiting roaduser, degradingfunction factor, build-up Q
                me[i][j] = new float[4];
                float m0 = bytema[i*2]+128f;
                float p0 = bytepa[i*2]+128f;
                float m1 = bytema[(i*2)+1]+128f;
                float p1 = bytepa[(i*2)+1]+128f;

                me[i][j][RUWF] = (m0+p0)/256f;
                me[i][j][DECF] = (m1+p1)/256f;     //  (between   0   and   2)
degrading factor
                        me[i][j][WAIT] = 0;                //
num_waiting
                        me[i][j][BUCK] = 0;                // build up Q,
aka bucket
                        }
                }
        }

/** Creates a new ACGJ3Individual, providing the reproduction genes from daddy and
mummy */
        protected ACGJ3Individual(byte[] ma, byte[] pa)
        {
                bytema = ma;
                bytepa = pa;
                createMe();
                wait = 0;
                move = 0;
```

```
}

/** Constructor for loading */
protected ACGJ3Individual ()
{
}

/** Resets the buckets and waiting values for this Individual */
protected void reset() {
        for(int i=0;i<me.length;i++) {
                for(int j=0;j<me[i].length;j++) {
                        me[i][j][WAIT] = 0;
                        me[i][j][BUCK] = 0;
                }
        }
}

/** Returns the genes of this ACGJ3Individual */
protected byte[][] getGenes() {
        byte[][] ret = {bytema,bytepa};
        return ret;
}

/** Returns some reproduction genes from this ACGJ3Individual. */
protected byte[] getReproGenes(float cross, float mutate) {
        byte[] ar1,ar2;
        if(random.nextFloat() > 0.5f) {
                ar1 = bytema;
                ar2 = bytepa;
        }
        else {
                ar1 = bytepa;
                ar2 = bytema;
        }

        /* Crossover? */
        int num_genes = ar1.length;
        for(int i=0;i<num_genes;i++) {
                if(random.nextFloat() < cross) {
                        int pos = (int) Math.floor(random.nextFloat()*8);
                        int temp1 = (int) Math.pow(2,pos)-1;
                        int temp2 = ar1[i]&temp1;
                        int temp3 = ar2[i]&temp1;
                        ar1[i] &= temp2;
                        ar1[i] += temp3;
                }
        }

        /* Mutation? */
        for(int i=0;i<num_genes;i++) {
                for(int j=0;j<8;j++) {
                        if(random.nextFloat()<mutate) {
                                ar1[i] ^= (byte) Math.pow(2,j);;
                        }
                }
        }
        return ar1;
}
```

/* Returns the fitness of the current Individual, as was set at iterating through this generation */

/* Returns the Fitness of this individual as calculated elsewhere */

```
        protected float getFitness() {
                return fitness;
        }

        /* Sets the fitness of this Individual  */
        /* Sets the Fitness of this individual */
        protected void setFitness(float fit) {
                fitness = fit;
        }

        /* Sets the number of cars that this Individual caused to wait  */
        /* Sets the number of Roadusers this Individual made waiting */
        protected void setWait(int _wait) {
                wait = _wait;
        }

        /* returns the number of Roadusers this Individual caused to wait */
        /* Returns the number of Roadusers I caused to wait */
        protected int getWait() {
                return wait;
        }

    /* Sets the amount of Roadusers that did move during this Individual's lifespan*/
        /* Sets the number of Roadusers this Individual made moving */
        protected void setMove(int _move) {
                move = _move;
        }

/* Returns the amount of Roadusers that did move during this Individuals lifespan/
        /* Returns the number of Roadusers I caused to move */
        protected int getMove() {
                return move;
            }

/* Calculates the current gain of the given TrafficLight  parameter : node the Id of the
Node this TrafficLight belongs to tl The position of the TrafficLight in the TLDecision[][]
* return    : the gain-value of when this TrafficLight is set to green
/* Returns the gain for the provided TrafficLight to be set to Green */
protected float getQValue(int node, int tl) {  int num_waiting =
tld[node][tl].getTL().getLane().getNumRoadusersWaiting();
                me[node][tl][2] = num_waiting;

                float oldQ = me[node][tl][BUCK];
                float newQ = 0;
                float ruW = me[node][tl][RUWF];
                float dFS = me[node][tl][DECF];
                float dec = 1;
                for(int i=0;i<num_waiting;i++) {
                        newQ += ruW*dec;
                        dec *= dFS;
                }
                if((oldQ+newQ)>=100000)
                        me[node][tl][BUCK] = 10000;
                else
                        me[node][tl][BUCK] = oldQ + newQ;
                        return newQ + oldQ;
            }

/* Empties the 'gain-value' bucket partly, which is being filled when Roadusers are
            * waiting/voting for their TrafficLight to be set to green.
            *
```

```
                * parameter :  node The Id of the Node of which a bucket has to be
emptied a bit, tl The TrafficLight of which a bucket has to be partly emptied in the
TLDecision[][] */
/* Empties the build-up Gain as a roaduser left the scene */
protected void relaxBucket(int node, TrafficLight tl) {
                int tli = -1;
                int num_tls = me[node].length;
                for(int i=0;i<num_tls;i++)
                        if(tld[node][i].getTL()==tl) {
                                tli = i;
                                break;
                        }
                float curWait = me[node][tli][WAIT];
                float curBuck = me[node][tli][BUCK];

                if(curWait>0) {
                        me[node][tli][BUCK] = curBuck*((curWait-1)/curWait);
                        me[node][tli][WAIT]--;
                }
                else {
                        me[node][tli][BUCK] = 0;
                        me[node][tli][WAIT] = 0;
                }
        }

/* When a Roaduser may drive according to it's Sign, but cant as the desired Drivelane
 is full, (some/all) of the current gain-bucket of it's current TrafficLight is siphoned over
to the desired Drivelane, making it more probable that that lane will move, making it
possible for this Drivelane to move.
 * parameter :  node the Id of the Node at which the Roaduser is waiting
 * sign The TrafficLight the Roaduser is waiting for
* des The TrafficLight that controls the traffic on the desired Drivelane    */
        protected void siphonToOtherBucket(TrafficLight here, TrafficLight there)
        {
                int hereId = here.getId();
                int thereId = there.getId();

                int hereNodeId = here.getNode().getId();
                int thereNodeId = there.getNode().getId();

                int hereRelId = -1;
                int thereRelId = -1;

                TLDecision[] hereltds = tld[hereNodeId];
                int num_hereltds = hereltds.length;
                for(int i=0;i<num_hereltds;i++) {
                        if(hereltds[i].getTL()==here)
                                hereRelId = i;
                }

                TLDecision[] thereltds = tld[thereNodeId];
                int num_thereltds = thereltds.length;
                for(int j=0;j<num_thereltds;j++) {

                        if(thereltds[j].getTL()==there)
                                thereRelId = j;
                }

                float buck = me[hereNodeId][hereRelId][BUCK];
                float curWait = me[hereNodeId][hereRelId][WAIT];
                if(curWait>0) {
                        if(me[thereNodeId][thereRelId][BUCK] < 100000)
me[thereNodeId][thereRelId][BUCK] += buck/curWait;     // Aka: make sure that lane
gets mmmmooving!
        }
                    else {

                         me[hereNodeId][hereRelId][BUCK] = 0;
                         me[hereNodeId][hereRelId][WAIT] = 0;
                    }
               }

               // XMLSerializable implementation of ACGJ3Individual

public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
               {      wait=myElement.getAttribute("wait").getIntValue();
                      move=myElement.getAttribute("move").getIntValue();
                      fitness=myElement.getAttribute("fitness").getFloatValue();
                      bytepa=(byte[])XMLArray.loadArray(this,loader);
                      bytema=(byte[])XMLArray.loadArray(this,loader);
                      me=(float[][][])XMLArray.loadArray(this,loader);
               }

               public XMLElement saveSelf () throws XMLCannotSaveException
               {      XMLElement result=new XMLElement("individual");
                      result.addAttribute(new XMLAttribute("wait",wait));
                      result.addAttribute(new XMLAttribute("move",move));
                      result.addAttribute(new XMLAttribute("fitness",fitness));
                      return result;
               }
```

```
		public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
		{	XMLArray.saveArray(bytepa,this,saver,"pa");
			XMLArray.saveArray(bytema,this,saver,"ma");
			XMLArray.saveArray(me,this,saver,"me");
		}

		public String getXMLName ()
		{	return myParentName+".individual";
		}

	public void setParentName (String parentName) throws XMLTreeException
		{	myParentName=parentName;
		}
	}

	public void showSettings(Controller c)
	{
		String[] descs = {"# of steps an individual may run", "Number of
Individuals in Population", "Crossover chance", "Mutation chance"};
		int[] ints = {NUM_STEPS, NUMBER_INDIVIDUALS};
		float[] floats = {CROSSOVER_CHANCE, MUTATION_CHANCE};
		etcSettings settings = new etcSettings(descs, ints, floats);

		settings = doSettingsDialog(c, settings);

		NUM_STEPS = settings.ints[0];
		NUMBER_INDIVIDUALS = settings.ints[1];
		CROSSOVER_CHANCE = settings.floats[0];
		MUTATION_CHANCE = settings.floats[1];
		setmapstructure(map);
```

```
    }

    // XMLSerializable implementation

    public void load (XMLElement myElement,XMLLoader loader) throws
XMLTreeException,IOException,XMLInvalidInputException
    {       super.load(myElement,loader);
            NUM_STEPS=myElement.getAttribute("s-num-steps").getIntValue();
            CROSSOVER_CHANCE=myElement.getAttribute("co-
prob").getFloatValue();
            MUTATION_CHANCE=myElement.getAttribute("mut-
prob").getFloatValue();
            NUMBER_INDIVIDUALS=myElement.getAttribute("num-
ind").getIntValue();
            num_step=myElement.getAttribute("o-num-steps").getIntValue();
            num_wait=myElement.getAttribute("num-wait").getIntValue();
            num_move=myElement.getAttribute("num-move").getIntValue();
            System.out.println("LoadingACGJ3:   num_wait:"+num_wait+"   num-
move:"+num_move);
            pop=new ACGJ3Population();
            loader.load(this,pop);
            ind=pop.inds[myElement.getAttribute("ind-index").getIntValue()];
    }

    public XMLElement saveSelf () throws XMLCannotSaveException
    {       XMLElement result=super.saveSelf();
            result.setName(shortXMLName);
            result.addAttribute(new XMLAttribute("s-num-steps",NUM_STEPS));
            result.addAttribute(new                    XMLAttribute("co-
prob",CROSSOVER_CHANCE));
```

```
		result.addAttribute(new					XMLAttribute("mut-
prob",MUTATION_CHANCE));
		result.addAttribute(new					XMLAttribute("num-
ind",NUMBER_INDIVIDUALS));
		result.addAttribute(new XMLAttribute("o-num-steps",num_step));
		result.addAttribute(new XMLAttribute("num-wait",num_wait));
		result.addAttribute(new XMLAttribute("num-move",num_move));
		System.out.println("SavingACGJ3:	num_wait:"+num_wait+"	num-
move:"+num_move);
		result.addAttribute(new XMLAttribute("ind-index",
					StringUtils.getIndexObject(pop.inds,ind)));
		return result;
	}

	public void saveChilds (XMLSaver saver) throws
XMLTreeException,IOException,XMLCannotSaveException
	{	super.saveChilds(saver);
		saver.saveObject(pop);
	}

	public String getXMLName ()
	{	return "model."+shortXMLName;
	}

	// InstantiationAssistant implementation

	public Object createInstance (Class request) throws
	    ClassNotFoundException,InstantiationException,IllegalAccessException
	{	if (ACGJ3Population.class.equals(request))
		{ return new ACGJ3Population();
		}
```

```
            else if (ACGJ3Individual.class.equals(request))
            {  return new ACGJ3Individual();
            }
            else
            { throw new ClassNotFoundException
             ("ACGJ3 InstantiationAssistant cannot make instances of "+request);
            }
    }

    public boolean canCreateInstance (Class request)
    {       return ACGJ3Population.class.equals(request) ||
            ACGJ3Individual.class.equals(request);
    }
}
```

Appendix B:
Gantt Chart

Task / Duration	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Requirements Gatherring	■	■			■	■						
Software Installation	■											
Algorithm Research		■	■									
Dry Run			■	■								
GUI & Interface Idea				■	■	■						
Coding												
Testing & Debugging						■	■	■	■	■	■	■
Performance												
Documentations												

Status : Project Completed

References

[1] R. S. Sutton and A. G. Barto, ***Reinforcement Leaning: An Introduction,*** The MIT press, Cambridge MA, A Bradford Book, 1998.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," ***Journal of Artificial Intelligence Research,*** vol.4, pp. 237-285, 1996.

[3] G.J. Tesauro, "Temporal difference learning and TD-Gammon," ***Communications of the ACM,*** vol. 38, pp. 58-68, 1995.

[4] Robert H. Crites and Andrew G. Barto, ''Improving elevator performance using reinforcement learning," in ***Advances in Neural Information Processing Systems,*** David S. Touretzlq, Michael C Mozer, and Michael E. Hasselmo, Eds. 1996, vol. 8, pp. 1017-1023,The MIT Press.

[5] "Packet muting in dynamically changing networks: A reinforcement learning approach,"in ***Advances in Neural Information Proceuing Systems,*** Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, Eds. 1994, vol. 6,pp. 671-678, Morgan Kaufmann Publishers.

[6] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," ***J. Phys. I France,*** vol. 2, p.p. 2221-2229, 1992.

[7] N. Findler and J. Stapp, "A distributed approach to optimized control of street traffic signals," ***Journal*** of ***Transportation Engineering,*** vol.118-1, pp. 99-110, 1992.

[8] K. Tavladakis and N. C. Voulgaris, "Development of an autonomous adaptive traffic control system," in ***ESIT '99 - The European Symposium on Intelligent Techniques,*** 1999.

[9] H. L. Liu, Jun-Seok Oh, and W. Recker, "Adaptive signal control system with on-line performance measure," in ***81st Annual Meeting of the Transportation Research Board,*** 2002.

[I0] K. K. Tan, M. Khalid, and R. Yusof, "Intelligent traffic lights control by fuzzy logic," ***Malaysian Joumal of Computer Science,*** vol. 9-2, 1995.

[1 1] J.H. Lee, K.M. Lee, K.A. Seong, C.B. Kim, and H. Lee-Kwang,"Traffic control of intersection group based on fuzzy logic," in ***Proceedings of the 6th Intenational Fuzly Systenis Association World Congress,*** 1995, pp. 465468.

[12] I. Rechenberg, "Evolution strategy: Nature's way of optimization," in ***Methods and Applications, Possibilities and Lirtiitations,*** Berg". Ed., 1989, pp. 106-126, Lecture notes in Engineering.

[13] H. Taale, Th. Back, M. Red, **A.** E. Eiben, **J. M.** de Graaf, and C. A. Schippers, "Optimizing traffic light controllers by means of evolutionary algorithms," in ***EUFIT'98,*** 1998.

[14] T. L. Thorpe and C. Andersson, "Traffic light control using sarsa with three state representations," Tech. Rep., IBM corporation, 1996. [15] Thomas Thorpe, "Vehicle traffic light control using sarsa," M.S. thesis, Department of Computer Science, Colorado State University.1997.

[15] Richard S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," ***Advances in Neural Information Processing Systems,*** vol. 8, 1996.

[16] Wankyoo Choi, Hongsang Yoo, Kyungsu Kim, Ilyong chung, and sungJoo Lee. A **traffic light controlling flc considering the traffic congestion.** *Lecture notes in computer science page* – 69-75 springer 2002

[17] Stephen Druitt, James laired and Duncan fraser . Edinburg city centre : A microsimulation case study http://www.sias.co.uk/sias/paramics/articles

[18] HOLLAND John H.. ***Adaptation in Natural and Artificial Systems.*** University of Michigan Press, Ann Arbor, 1975.

[19] COELLO Carlo A. and PULIDO Gregorio T. ***Multiobjective Optimization using a Micro-Genetic Algorithm.*** In SECTOR Lee *et al.*, editor, *Proceeding of the Genetic and Evolutionary Computation Conference*, San Francisco, California, USA. Morgan Kaufmann Publishers, 2001.

[20] City Traffic. http://www.citytraffic.de/info_eng.pdf

[21] Handset helps to unstuck jams. http://news.bbc.co.uk/2/hi/technology

[22] Stephen Druitt-Some real applications of paramics microsimulation .

[23] 2004 IEEE Intelligent Vehicles Symposium University of Parma Parma, **Simulation and Optimization of Traffic in a City** Marco Wiering, Jilles Vreeken, Jelle van Veenen, and Arne Koopman.