Politehnica University of Bucharest

Computer Science Department

COLLABORATIVE ROUTE PLANNING IN VEHICULAR AD-HOC NETWORKS

Student:

Diaconescu St. Raluca

Advisors:

Prof. Valentin Cristea

Prof. Liviu Iftode

Abstract

Finding the best route towards a destination in the complex and complicated roads network of the modern cities is a major challenge in the attempt to improve traffic conditions. This project presents an application that offers a solution to this problem, based on the collaboration between drivers and the exchange of information between cars equipped with devices with short-range wireless communication capabilities. This application aims at reducing the total travel time towards a destination by providing the best route, dynamically computed using real time information about congestions and road conditions. This program comes as an extension for the "TrafficView" platform for inter-vehicle communication and was tested using a vehicular traffic simulator developed in collaboration with two other students. First, I give a general presentation of the simulator we have created. Second, I describe the algorithms used in computing a route and then the two mechanisms of dynamically route planning. Last, the results obtained during simulation are presented.

1. Introduction

One of the most challenging problems modern cities face today is represented by the large number of cars traveling on their streets daily. The constantly increasing vehicular traffic is the main cause for traffic jams and air pollution, a permanent source of stress and a factor contributing to the development of some health problems. Along with automobile accidents, they are two of the leading causes for decreased standard of living and loss of productivity. Due to the increasing number of people commuting daily between their homes and their jobs, primary roads and highways are also seriously affected.

Municipalities and transportation authorities are making desperate efforts to speed up the traffic flow. The solutions that they currently rely on are infrastructure dependent. Some examples include cameras at intersections or sensors in the road that send information to a centralized authority. Drivers that want to find information about traffic conditions have to connect to this authority and download the information. Such solutions are very expensive to install and to maintain and offer no scalability or flexibility.

Intelligent Transportation Systems (ITS)[1] are trying to solve these problems, aiming at relieving congestions, improving safety and enhancing productivity by developing new features and functionalities for individual vehicles. By equipping cars with wireless communication devices they can form ad-hoc networks (VANETS), in which Inter-Vehicle Communication (IVC) opens new perspectives for a broad range of applications. By gathering and disseminating information from and to other vehicles, this cooperative platform may provide many services to a driver. These services may refer to safety applications, like various warnings (brake, ice on road, intersection violation etc.), cooperative collision avoidance or mitigation, to applications concerned with operation and maintenance, like dynamic route planning, transmitting information on weather

conditions, and commercial applications like electronic payments, reservations, advertisements or file transfers.

Finding the best route towards a destination in the congested roads network is a major challenge in the attempt to improve traffic conditions. By avoiding routes that are already jammed the driver considerably reduces the total trip delay and helps by not increasing the density of cars in already saturated streets.

Modern road networks are very complex and complicated making the search for the best route difficult. Shortest path algorithms generally produce routes that are not suitable for human drivers while man made decisions many times prove to be inefficient and expensive because they result in excessive travel time. All these decisions are based on static information about roads and do not take into consideration what actually happens at the current time on those roads.

This is the case of GPS navigators currently on the market. They provide tools for route planning and route guidance but their decisions are based on static information.

Some solutions to this problem are based on the information received as an input from the driver. They use problem solving experience gained from previous trips to determine the best route for the diver [2]. Learning from experience just solves a small part of the problem. Even if it can be determined from many trials that some roads get jammed during some time intervals in particular week days, it does not take into consideration unpredictable events like accidents or unfavorable road conditions like ice. It also raises problems when traveling in unfamiliar territory but this can be tackled by learning from other drivers' experience.

Other solutions use the information obtained from infrastructure to dynamically generate the best route [7]. But they can be applied only for the roads that are monitored by previously deployed sensors (loop detectors in asphalt, cameras in intersections) and that allows cars to access this data.

This paper presents a two solution to this problem, based on the collaboration between drivers and the exchange of information between cars. Assuming that cars are equipped with wireless communication devices, with processing power and with GPS devices, they can collaborate in order to exchange information about traffic conditions and to dynamically determine which the best route towards a destination is.

The collaborative based route planning systems presented here are intended as an extension for "TrafficView" [5], an IVC platform that can be embedded in the next generation of vehicles to provide the drivers with a real-time view of the road traffic far beyond what they can physically see.

One application begins by statically computing a route towards a destination, based on a set of parameters associated to the roads: lane number, road type (speed limit) and distance. For this an efficient searching algorithm was used [2]. It then sends interrogations about traffic conditions on the roads it will travel and recalculates the route based on information received from other drivers (average speed, weather conditions or accidents). The other application is based on a hybrid mechanism formed by both infrastructure nodes and vehicles that exchange information so that cars can be routed by fixed infrastructure nodes on the best way towards their destination.

These applications aim at reducing the total travel time towards a destination. The performances of these applications are studied by comparing the travel time obtained when using these applications with the duration of the same journey when only shortest path algorithms and static information are used in the route's computation.

The applications were tested using a traffic simulator developed in collaboration with two other students [3]. This is a java based discrete event simulator that runs a microscopic traffic simulator and a wireless communication model.

2. Simulation Environment

2.1. TrafficView Platform

The collaborative routing application comes as an extension to the "TrafficView" platform. This is a framework to disseminate and gather information about vehicles on the road. Using such a system, a vehicle driver becomes aware of the road's traffic, which helps driving in situations with low visibility like foggy weather and creates a platform for transmitting information about road conditions like accidents or jams.

TrafficView is an IVC system deployed over an ad-hoc 802.11 wireless network of vehicles. Its goals are to develop a network stack that can face the challenges of high mobility of nodes and frequent disconnections of topology and to create tools that can accurately model real traffic conditions.

Each car is equipped with a GPS receiver, with a short-range wireless communication device, with processing power and a display unit. The "TrafficView" application obtains the car's position from the GPS device and using a digital map, constructed from TIGER files offered by U.S. Census Bureau [15], it positions itself on a road, more precisely on a road segment. The accuracy of the GPS information is of a few feet in good weather conditions but it can decrease when used near tall buildings or in forests. Some other sources of error include ionosphere and troposphere delays, signal multi-path, receiver clock errors, orbital errors, number of visible satellites, satellite geometry/shading or even intentional degradation of the satellite signal. This is why a positioning on a road lane is still not possible. The GPS signal is also used to obtain time information, synchronizing all cars.

After gathering this information, the car broadcasts it at regular time intervals. When receiving information about other cars, this data is added to the one about the current car and broadcasted at the next step (Figure 1).

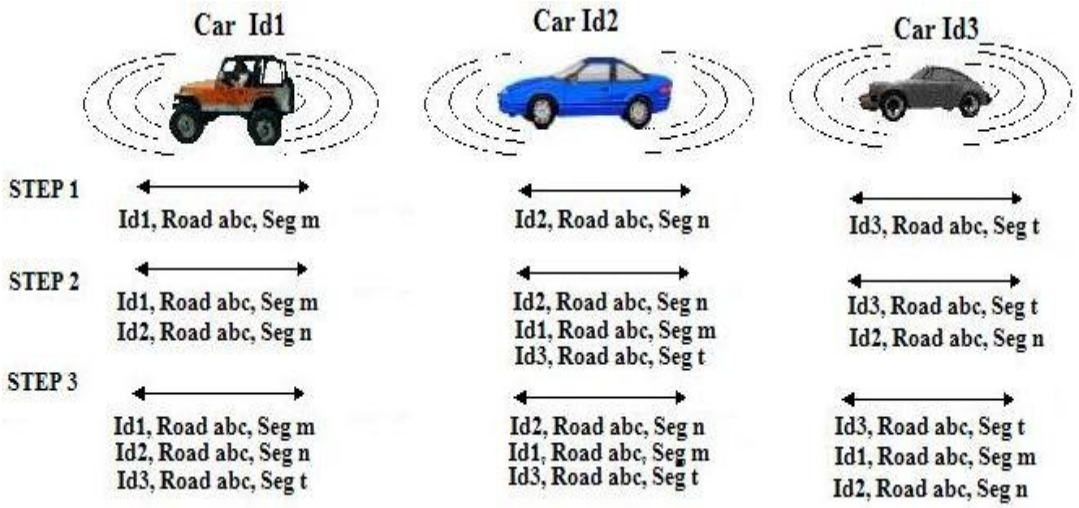


Figure 1. Information dissemination

Using 802.11 for inter-vehicular communication raises a number of problems. Due to the limited bandwidth only information about a relevant area should be kept and disseminated. This technology has a range of about 30-100 m outdoors, but it can be extended by adding a signal amplifier so a maximum 500 m range can be achieved.

All the information a car has is also displayed to the driver. Figure 2 is a screenshot from a test run in Politehnica University Campus. The driver is presented with a static image of the map on the right (as a classic GPS navigator) and on the left with a dynamic image of the road formed from the information received from other cars in vicinity. This is why it can be considered a Dynamic GPS.

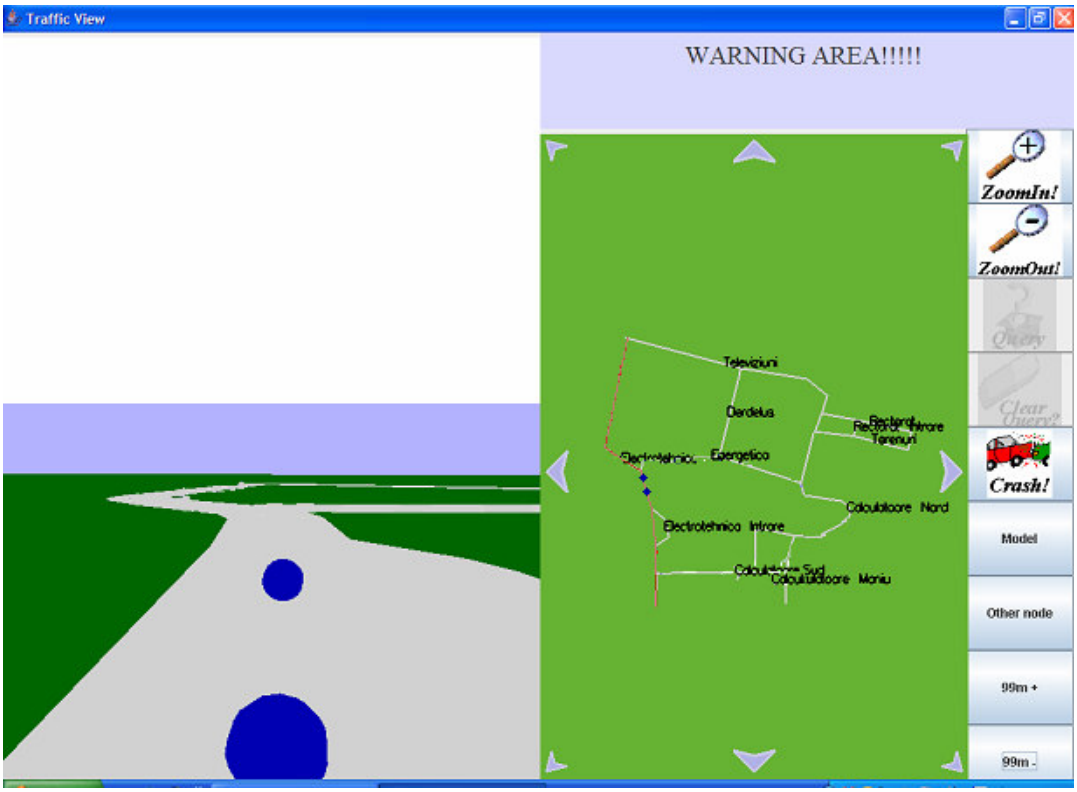


Figure 2. Screen shot from a test run in Politehnica University Campus

The application was developed in Java, using OpenGL for the graphical user interface.

Each car has a local database containing information about other cars. Every record from this database is periodically updated or deleted if it gets old. All records have the following fields: Identification (used to uniquely identify the record corresponding to a vehicle), Position (the position of the vehicle – latitude & longitude), Speed (speed of the vehicle), Original Timestamp (the time this record was created) and Receive Timestamp (the time when the record was received).

This database is shared by four components:

- GPS module: each second the current car's position is updated and the corresponding record is modified.

- Sending module: all the information a car has is periodically broadcasted.
- Receiving module: listens to messages broadcasted by other cars and updates the information in the database.
- GUI module: displays the map and the information in the database to the driver.

Using the basic functionality provided by TrafficView, other applications have been developed in order to provide even more services to the driver. A query-reply protocol (VITP) [8] for the vehicular ad-hoc network was implemented, allowing the driver to initiate queries, asking for specific information like the average speed on a road segment.

The Vehicular Information Transfer Protocol (VITP) [8] is an application-level protocol which uses the ability of the VANET to transport messages. It defines two types of messages that can be initiated by a car: POST and GET. The POST message is a packet that will be routed toward a destination and will be periodically broadcasted by the vehicles in that region. Its purpose is to inform drivers in an area about particular conditions present in that zone. The GET message is a packet that tries to find out some type of information about a destination area and that will return an answer to the vehicle that generated it. Routing a packet towards an area of interest raises a number of problems due to the fact that VANETs are not IP based networks. VITP tackles this problem by using a geographical routing based on streets' topology and on distance.

The first application presented in this paper uses the GET mechanism implemented by VITP to obtain the needed information about road conditions and traffic in order to dynamically compute the routes. The other one uses simple broadcasted messages for communication.

2.2. Traffic Simulator

2.2.1. Overview

When developing a vehicular application, a very complex and difficult step is the testing process. Accurate testing involves thousands of nodes, but performing actual experiments is burdensome even for just a few nodes. This is why a simulation tool is needed. For simulation of vehicular ad hoc networks two different aspects have to be taken into consideration: a network simulator, capable of simulating the behavior of a wireless network, and a vehicular traffic simulator, able to provide an accurate mobility model for the nodes of a VANET.

Recent studies [16] have proven that the vehicular mobility model is very important, and in order to obtain relevant results, the two components should be integrated. If an inaccurate mobility model is used, like the popular random waypoint model (which may work for some mobile ad-hoc networks, but is definitely not an accurate representation of mobility in a VANET), false results can be obtained [16]. Integrating the two components is especially important for the situation when the vehicles' mobility is influenced by the messages the nodes receive. For instance, if some nodes are to change their route as a reaction to messages they receive, we cannot use previously generated vehicular traffic traces.

This is why we have chosen to develop our own simulation tool, comprising the 2 previously mentioned components: a microscopic traffic simulator, and a wireless communication model. The entire project was implemented in Java.

2.2.2. Related Work

Existing network simulators deal with issues like medium access control, signal strength, propagation delays.

One of the most popular ones is NS-2, which is a discrete event simulator. It is probably the most widely used simulator in ad hoc networking. It provides support for TCP, routing, multicast, propagation models in wired or wireless networks and new protocol layers can also be added. It was developed by the "Information Sciences Institute" – University of Southern California [9].

Another interesting simulator is Jist/SWANS, developed by a team at Cornell University[10]. JiST is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. SWANS is a scalable wireless network simulator built on top of the JiST platform. The authors claim that "SWANS can simulate networks that are one or two orders of magnitude larger than what is possible with GloMoSim and ns2, respectively, using the same amount of time and memory, and at the same level of detail."

However, a general-purpose wireless network simulator is by no means enough for an accurate simulation of a vehicular network. Nodes in a wireless network usually move according to the random-waypoint model. That means they have an origin and a destination and move towards the destination. But vehicles only move along roads and that is a very particular situation. Furthermore, real vehicles move according to very particular traffic models, due to the street topology, intersections, and, of course, drivers' behavior. That takes us to the second very important aspect of a vehicular network simulator, which is simulating a mobility model as close as possible to real vehicular mobility.

Vehicular traffic simulators can be classified in macroscopic and microscopic simulators. Macroscopic simulators deal with flows of vehicles, while microscopic simulators take into account the movement of each particular vehicle.

There are a lot of vehicular traffic simulators, but many of them are commercial. Of course, these simulators have not been designed especially for vehicular computing. They are primarily used to validate projects, like building a new road, or a new tram line, or for designing effective traffic signals.

An example of a commercial vehicular traffic simulator is VISSIM [11]. It is a microscopic simulator and implements driver behavior models, like car-following or lane changing, and, according to its producers, it is used in over 70 countries.

A very used simulator is CORSIM, developed by the McTrans Center, University of Florida, which can be purchased for about 500$ [12].

A very interesting simulator was written by a team at Northwestern University. It is based on a vehicular traffic model designed by them, called Street Random Waypoint (STRAW). Their simulator is implemented on top of JiST/SWANS, and it is free and open-source [13]. They have used the simulator in order to prove that studying routing protocols for a vehicular network without an accurate vehicular traffic model is wrong. To do that, they compared results obtained with the Random Waypoint model (which is a very inaccurate representation of a vehicular network) with results obtained with their complex model (STRAW). The experiments clearly indicate that using the Random Waypoint model will not produce accurate results for a vehicular network [16].

2.2.3. Our Simulator

We have created an integrated simulator that combines the two features described earlier: an accurate vehicular mobility simulator, based on validated driver behavior models, and a wireless network simulator. It is a java based discrete event simulator, with an OpenGL interface.

2.2.3.1. Scenarios Generation

The simulator uses real digital maps, TIGER files [14], offered by U.S. Census Bureau. In particular we use ".RT1" and ".RT2" files that give information about roads' name and type, about the latitude and the longitude of the beginning point and the end point, but also the coordinates of some intermediary points. By parsing the information contained in these files, we create a Java object containing all the information. Furthermore, some improvements are also added to the map: roads are split in even smaller road segments by adding more intermediary points and roads with the same name that are connected are merged if they don't form a loop.

Because in the TIGER files no information is given about traffic lights, after the map is created, for each cross traffic lights or just priority signs are assigned in a heuristic manner. If all roads in a cross are of the same type (major or secondary), a traffic light will be created in the cross. If roads have different types, major roads will be given priority signs and secondary roads will have to give priority. For very dense city scenarios it turned out to be better to put traffic lights in all intersections in order to give a chance to all cars entering to pass through the intersection because in this type of scenario cars coming on roads that have to give priority frequently got stuck in the intersection waiting for the others to pass and having no space to cross the intersection.

All this information is stored in a Java object, called "Map". The object will be serialized in a file, so that all these computations will not be performed again. We have designed a simulator module which allows the user to interactively create various traffic scenarios. This module is made of 2 sub-modules.

The first one will load an existing Map object (from a file) and add a list of entries and exits. Various routes are computed between each entry-exit pair. All this information will be stored in a Java object and serialized in a file - "Scenario

Map File". This object is in fact a blank traffic scenario that will be later configured and loaded in the simulator.

The next module will load this object from a "Scenario Map File" and provide the user with a GUI which can be used to add traffic information on the map. The user can specify flows of vehicles and the routes to follow between entries and exits. All the information is stored in another Java object and serialized in a file – "Final Scenario File". This file contains all the information required by the simulator in order to run a specific scenario. In the following paragraphs, I give details about the GUI that can be used in order to create a traffic scenario.

Figure 3 presents the first module that allows the user to select the desired TIGER files and to create a "Map" object and a "Scenario Map File".

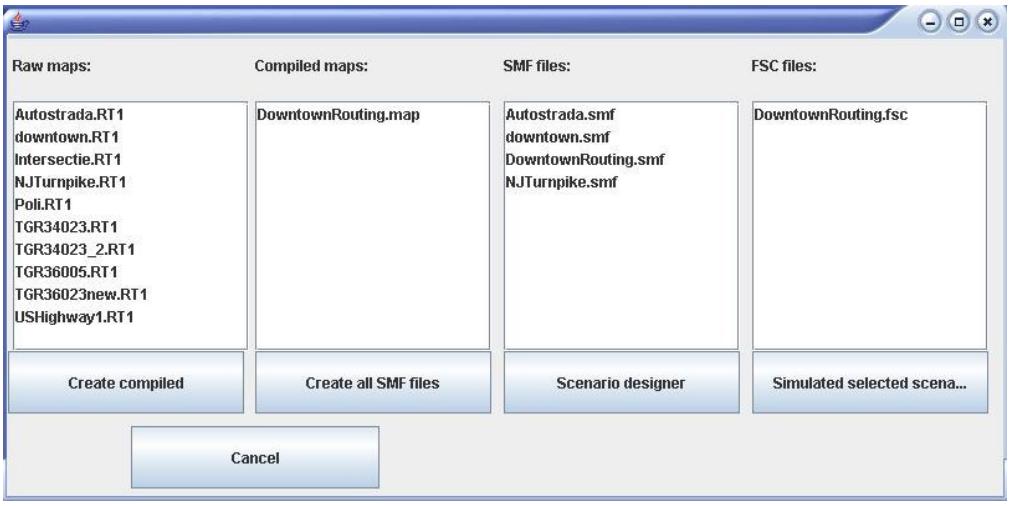


Figure 3. First configuration window

After creating a map from the TIGER files, the next step is to start the "Scenario designer" and to configure the desired model of traffic. First, the

scenario selection frame (Figure 4) allows the user to select a specific scenario and to configure it from the beginning or to load a previously configured scenario and to modify it. The specific scenarios are those which can be found in the “Scenario Map Files”. These files are placed in a specific directory and the scenario designer will look for all of them, and allow the user to choose which one to use.

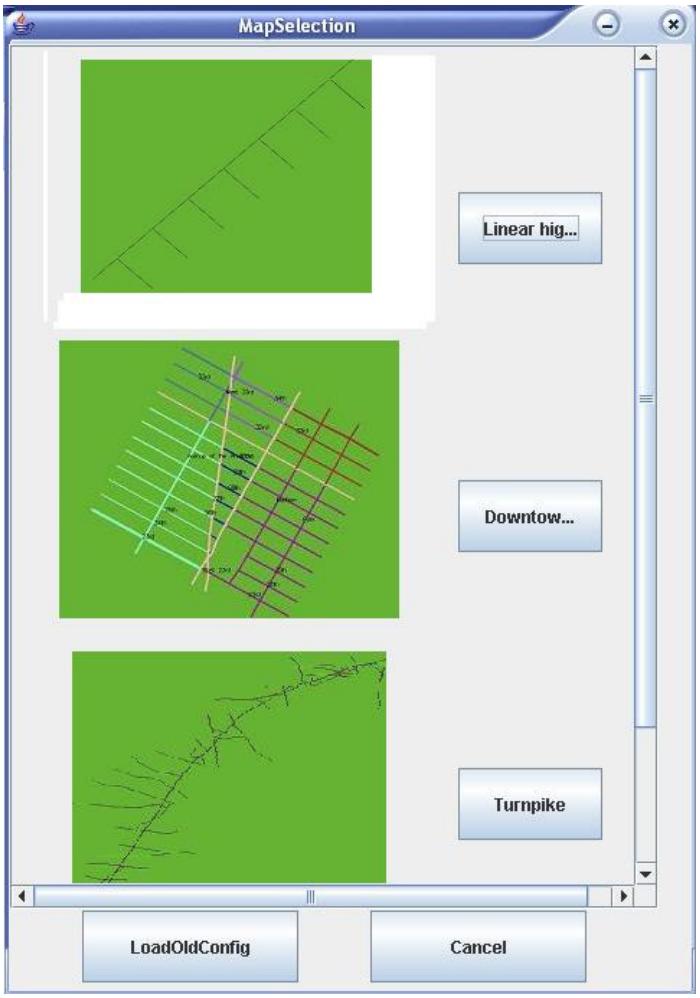


Figure 4. Scenario Designer- selection of the scenario

After selecting a scenario, the configuration phase begins (Figure 5). The user is presented in the upper left corner with all the available entries. For each

entry, a list of exits towards which a route exists is displayed. The user can select a route and on the map in center the route will be displayed with red. For each entry the user wants to use, the percent of **vehicles/hour/lane** has to be specified in the first text field from the right upper corner. The distribution of driver personalities can be configured from the next text fields. Three types of personalities have been implemented: **very calm, regular** and **aggressive**. The default distribution implies equal values.

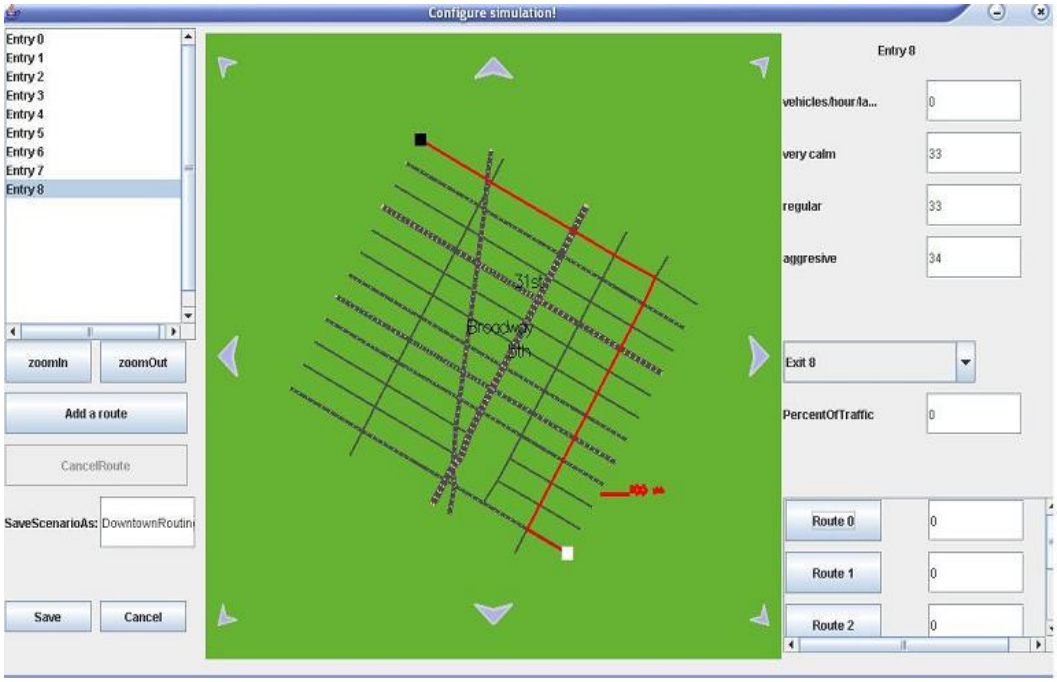


Figure 5. Scenario Designer- scenario configuration

From the total flow of vehicles starting from one entry point, the percent of traffic heading towards each exit should be specified. For the traffic starting from one entry toward an exit, the percent following each route should also be specified. All these values should not necessary add up to 100. After the configuration is over, the corresponding values are added up and the percent of each one is computed.

Because the routes pre-calculated in the "Scenario Map File" may not be enough, the user can defined its own routes. To add a new route, and implicitly the corresponding entry and exit, the user has to first press the "Add a route" button. Then it can start selecting points from the map. The first should be the entry point. Next, only the crosses where the road changes should be selected. Finally the exit point should be selected. All these points will be saved and when the user will press the "Add it" button, the route will be checked for validity and added to the scenario. The user will then be able to configure the previously described parameters.

The configured scenario can be saved under a name specified by the user. If no name is specified, the default name used is the name of the map.

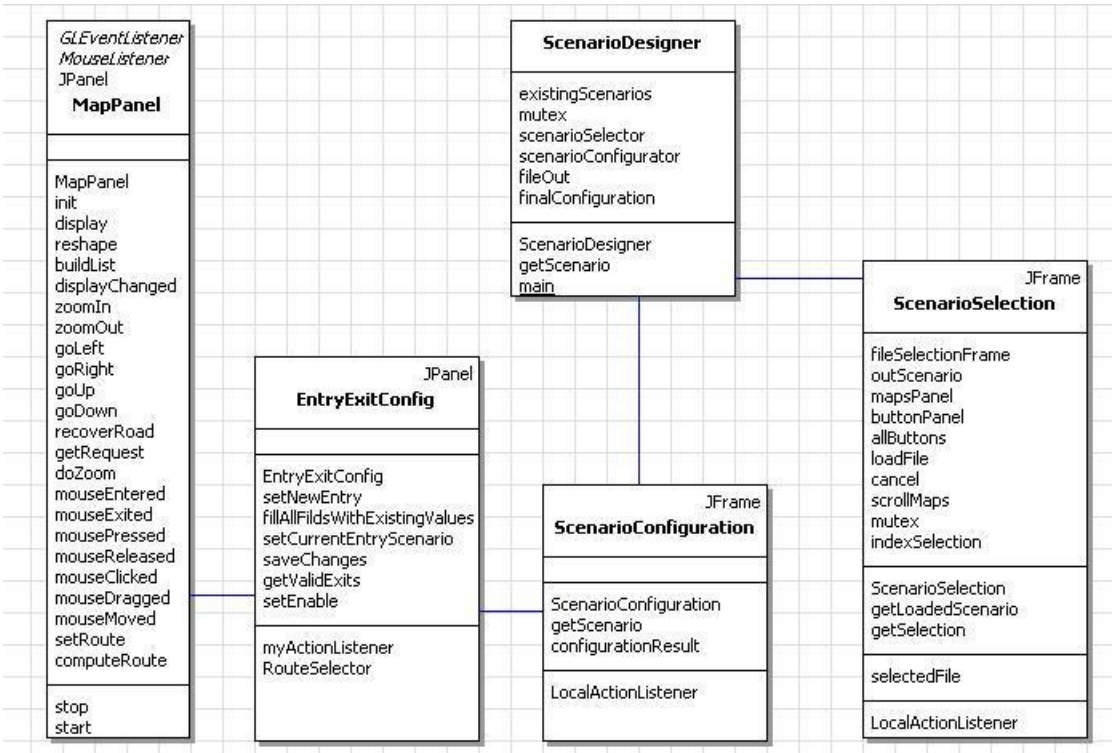


Figure 6. UML diagram of the scenario designer

Figure 6 presents the class diagram for the GUI of the scenario designer

The first method that is being called is the “main” method from the “Scenario Designer” class. This method will search through the “./maps/smf” directory for blank scenarios (“Scenario Map Files”). It loads all scenarios found and displays the “Scenario Selection” frame. From this frame the user can select a specific scenario to configure from the beginning or select a previously configured scenario and to change some parameters. After a blank scenario or a previously configured one has been selected, this structure is delivered as an input parameter for the “Scenario Configurator”.

The “Scenario Configurator” is the container that manages all the configuration components. It encloses a list of all existing entries, a set of buttons that control the map (zoom in, zoom out), the buttons for saving or canceling the current configuration, the buttons that set the configuration mode (adding a route or setting an entry’s parameters) and an “Entry Exit Configurator” which is responsible with displaying all configuration parameters corresponding to the currently selected entry and with displaying the map.

The “Entry Exit Configurator” displays all the exits to which a route exists from the currently selected entry and all the parameters associated with an entry: vehicle flow, driver types and percent of flow for each exit. For each pair entry-exit, a list of routes is presented. For each route the percent of traffic that follows that route out of the total traffic between that entry and that exit has to be specified.

The “Entry Exit Configurator” is also responsible with displaying the map. On this map, the current entry is shown as a white square and the current exit is displayed as a black square. Each route between the two can be shown by a red line when the corresponding button is pressed. The map is also in charge with collecting the points provided by the user as coordinates for a new route.

Figure 7 presents the class diagram for the structures used by the scenario configurator.

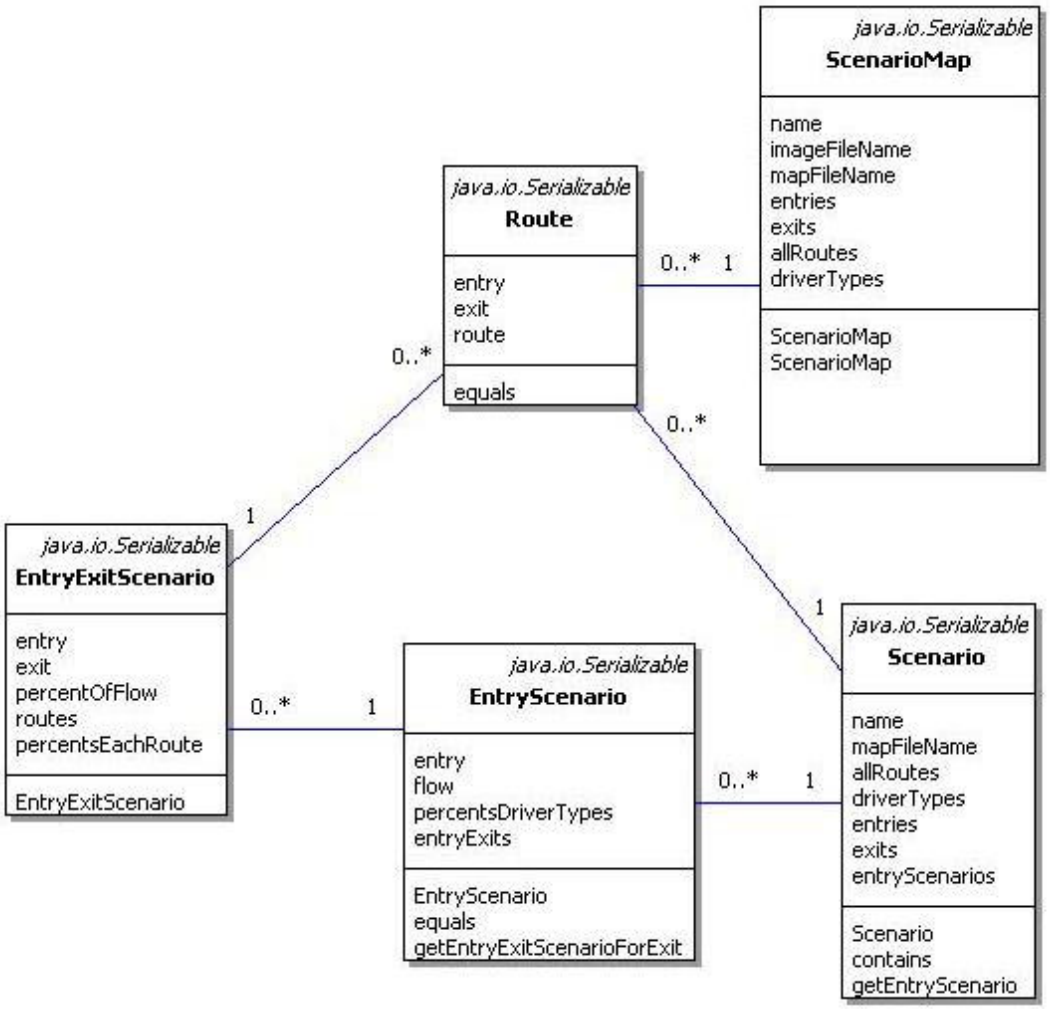


Figure 7. UML diagram of the scenario designer

A blank scenario is described by the "Scenario Map" class. For each scenario the following are kept: the scenario's name, the name of a picture that presents the map, the name of the map used, a list of driver types, a list of entries, a list of exits and a list of routes. Each route is described by an entry, an exit and a

list of segments that form the route. A “Scenario Map” object is given as an input parameter to the scenario designer. The designer produces as an output a “Scenario” object.

The initial purpose of the “Scenario” object was to keep only information about the configured routes and to keep it in a manner easily accessible for the simulator. This was done by keeping a list of “Entry Scenarios” for all entries that were configured. Due to the fact that many times it is useful to take a previously completed scenario and just tune it, it proved necessary to keep also the list of all entries, exits and routes from the “Scenario Map”.

An “Entry Scenario” stores all the parameters corresponding to an entry: the flow of vehicles entering the map through that point, the proportions of driver personalities, and, for each configured exit, an “Entry Exit Configuration”.

The “Entry Exit Configuration” defines the traffic between an entry and an exit. It keeps all the routes used by the vehicles traveling between the two points, the percent of traffic distributed to each route and the percent of traffic coming towards that exit from the total traffic entering the current entry.

2.2.3.2. Graphical User Interface

For the developed simulator, a graphical user interface was created. By means of this interface the user can visualize the simulation scenario and as experience proved it, it helps identifying particular cases that otherwise would be much harder to detect.

The GUI is made up of three components: a representation of the map and of the cars that are currently simulated, an area for controlling the simulation and an area where statistics are displayed (Figure 8).

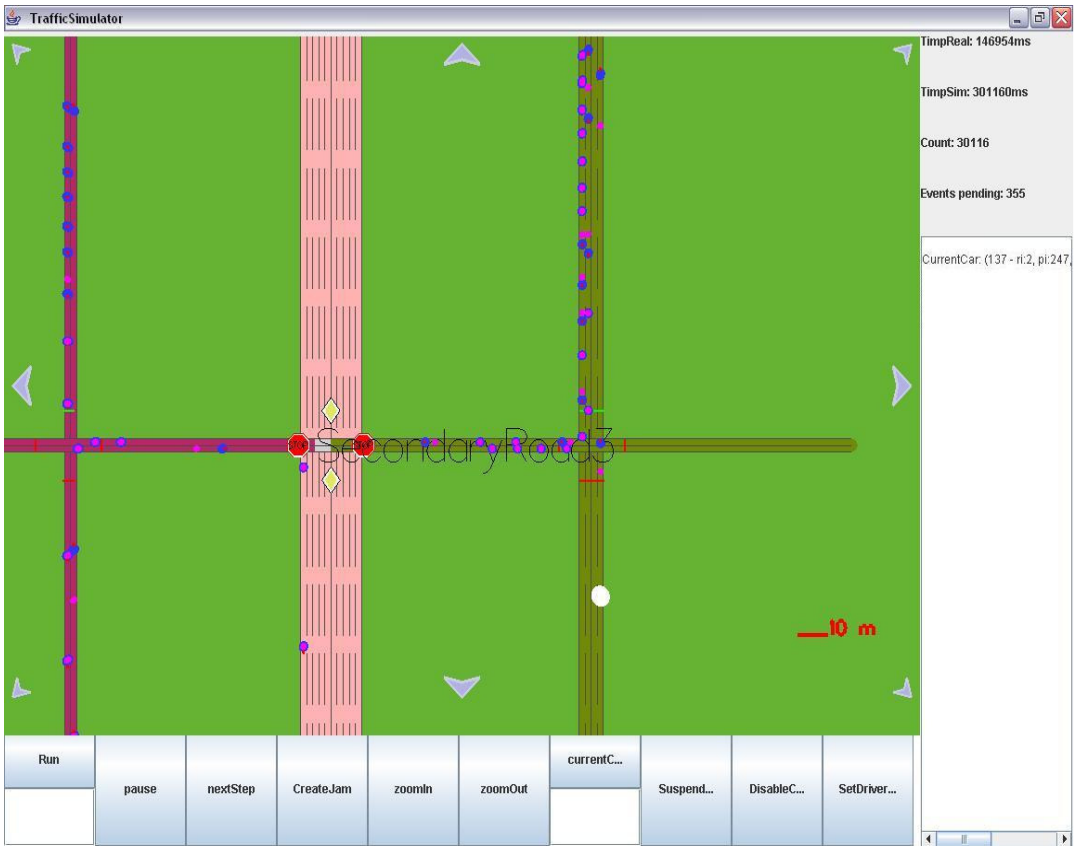


Figure 8. Simulator's graphical user interface.

The map representation allows the user to move the visual field in any direction by using the arrows drawn on the map's borders. The user can also select to zoom in or to zoom out by means of two buttons. The visual field can be directly moved to a specific area by pressing the mouse button in the upper left corner of the specific area, dragging the mouse and releasing the mouse button in the lower right area of the desired area.

Cars are displayed as spheres and for each car two spheres are displayed. The simulator generates cars that move like real cars, simulating a continuous motion (a high time resolution). On the other side, the simulated cars receive their position by means of the GPS device once a second (a lower resolution motion) This is why

two spheres are displayed: the blue one displays the car in the more discrete movement, as it positions itself once every second using the information received from the GPS device, while the red smaller sphere displays the real position of the car, as it is moved much more frequently by the simulator (simulating a continuous motion).

The user can select a car from all the cars displayed and, by doing this, some information about that car will be shown. The selection of a car is done by clicking twice on the map. The first click will zoom in closer to the area around the car, for better accuracy, and the second will select the car.

The car that has been selected is drawn as a white sphere and all the cars it knows about will be displayed as purple spheres. This is very useful when the user wants to visualize exactly how much information a car has about other traffic participants. Also, on the right side of the map, information about that car's position is displayed.

Once a car has been selected, the user can choose to switch to a different view. By pressing the "SetDriverView", the user can see the image that would be displayed to the driver in the selected car (Figure 9). This image is constructed from the information a car has about other cars around it.

This perspective is particularly interesting because it can provide the driver with traffic information that can be of real help in situation characterized by low visibility due to weather conditions or during night time driving.

In this perspective the user can also see the traffic lights for the surrounding area. This can be done for the simulator because this information is available directly from the simulator's engine. In real traffic conditions, such information would not be available unless traffic lights were equipped with wireless communication devices and would exchange information with cars approaching the intersection.

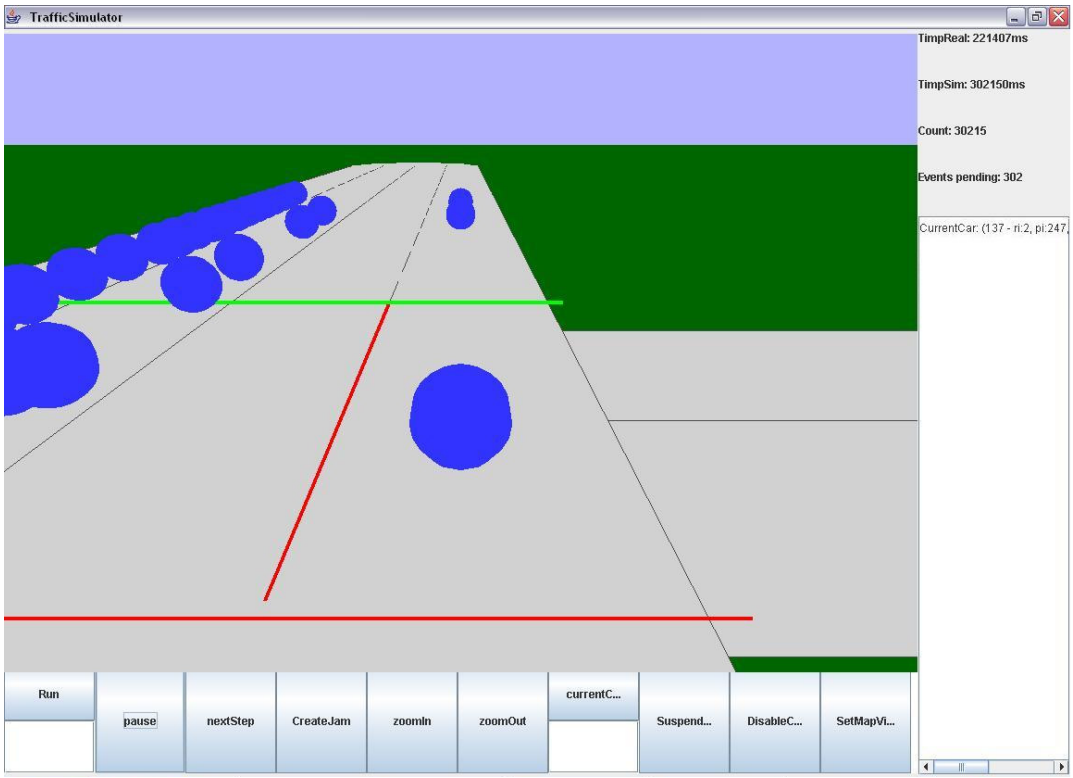


Figure 9. Simulator's "DriverView" perspective

In this view, the button that was used to change to the "DriverView" perspective is now used to change back to the map representation perspective.

In the upper right corner the user is displayed some information about the simulation development: the time elapsed from the beginning of the simulation, the time that has been simulated, the number of discrete time periods that have been simulated and the current number of events awaiting to be simulated.

The user has full control over the simulation. He can choose to execute just one time interval by pressing the "NextStep" button or it can choose to continuously execute all events by pressing the "Run" button. After the "Run"

button has been pressed, the user can pause the simulation by pressing the “Pause” button.

The user can also choose a car by its id. Under the “selectCar” button there is a field where the id can be typed and if a car exists with that id, when the “selectCar” button is pressed, that car will be selected and information about it will be presented in the same way as when selected by a mouse click.

Although the graphical user interface is very useful for observation about simulation development that would be otherwise very difficult to make, it is very consuming in terms of resources. This is why the “SuspendGUI” button was introduced. If the user wants to let the simulation advance faster for a period, by means of this button it can interrupt the display and allow the simulation to proceed more rapidly.

Some times only the mobility of cars may be of interest. In these situations, simulating the communication induces an unnecessary overhead so the “Disable Communication” button allows the user to skip the simulation of inter vehicular communication.

Figure 10 presents the UML class diagram for the graphical user interface.

The “Display” class describes the container for all the other components. It handles the creation of all the other panels and it handles the exchange between the map perspective and the driver perspective.

The “Controls” Java object is the one that manages the interaction between the GUI and the simulator’s engine. It contains all the buttons that control the simulation flow (Run, Pause, NextStep, SuspenGUI, DisabeleCommunication) and also the buttons that control the map (ZoomIn, ZoomOut ,SetCurrentCar).

The “Statistics” panel handles the presentation of data about the simulation progress. This data is constantly updated by the simulator’s engine at every time period.

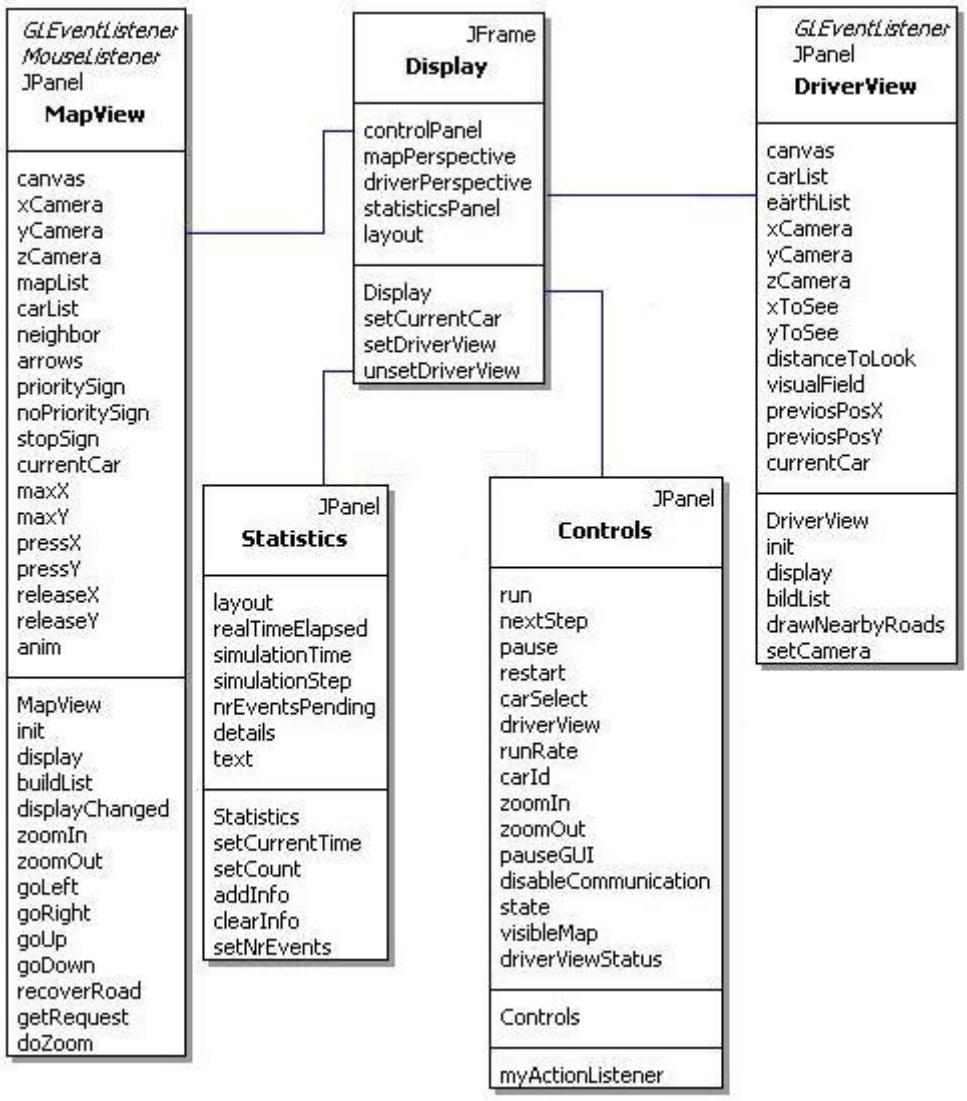


Figure 10 UML class diagram for the simulator’s graphical user interface

Both the “MapView” and the “DriverView” contain an OpenGL canvas on which the images are drawn. The “MapView” object computes in the beginning the representation of the current map and for every redisplay it just repaints this

image. For every redisplay, each car's position is recomputed and each traffic light's state is inspected before repainting. The "DriverView" displays only the road on which the current car is positioned and the roads with which this road has crosses.

3. Dynamic Route Computation Using a Greedy Approach

Existing implementations for a dynamic route planner rely only on the information received from previously deployed infrastructure (sensors in the road and cameras at intersections). They are not able to offer a complete solution because they can only provide information about the roads that are being monitored.

This project presents a solution based on the fact that cars are everywhere on roads and that information about road conditions can be exchanged between drivers. A car begins by statically computing a route towards its destination. Then the car begins its journey and using the "query-reply mechanism" described earlier in this paper, it periodically performs queries about traffic conditions on the roads it will travel. If accidents, jams or other traffic disturbances are reported, a detour will be computed based on this new information.

This chapter will present first the method used for finding a static route and then the algorithm used to dynamically determine information about traffic conditions on the roads ahead.

3.1. Map Splitting

Searching for a route is in essence a classic shortest path problem. Often classic algorithms are used to solve this problem, like Dijkstra or A*. But when these algorithms are applied to the vast road network they are extremely inefficient in terms of computation because many irrelevant zones are analyzed. Moreover the result is usually not the best solution for human drivers because people prefer traveling on major roads, while the shortest path most often comprises many secondary roads.

One efficient solution to this problem [2] uses some observations about the road network and the behavior of drivers in order to reduce the area searched just to areas of particular interest.

First, by dividing the roads into two categories, large roads (main roads and highways) and small roads (secondary roads), any map becomes naturally partitioned by the big roads in many smaller areas. These smaller areas contain secondary roads and are generally surrounded by large roads. In Figure 11, Bucharest's map illustrates this principle.

Second, human drivers' behavior studies [4] have proven that we prefer traveling on major roads.

The algorithm used to calculate a route is based on the above statements.

Figure 11. Bucharest City Map - main roads drawn with thick yellow lines partition the map in many small areas.

First, the big map is split in many smaller areas. The division process begins by identifying the major roads. For each major road, a digit is assigned in an area code. Each secondary road, more exactly each point of a secondary road will have an area code. The digits in the area code will indicate the relative position of the secondary road compared to a direction assigned to every main road. For example, if a secondary road is in the right of main road j, then the digit j in the secondary roads' code will be 0, and if in the left of the main road, the digit will be 1.

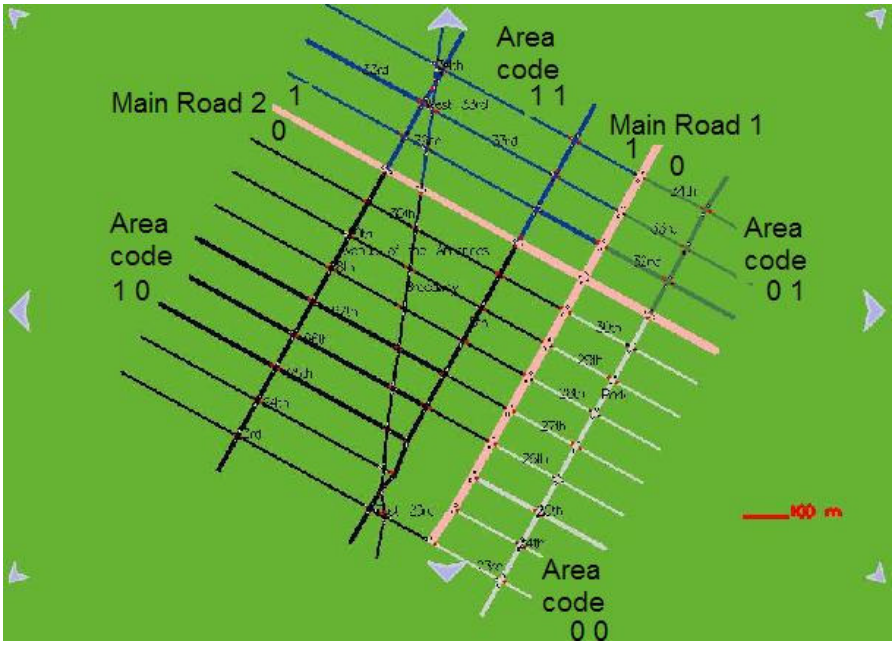


Figure 12. Two main roads dividing a map in four smaller areas

The process of forming the area codes begins by initializing every digit with "-1". Next, major road are assigned arbitrary a direction. Then every major road begins to mark with "0" on its corresponding position in the area code the road which cross it in the right and with "1" the ones which cross it in the left. This value is then propagated one secondary road point at a time, until a point that has already been assigned a value for that digit is encountered. When the division

process ends, each point should have an area code associated, code formed of “0”s and “1”s. Main roads will also have a similar code, only that on the position corresponding to them they will have “2”, marking that they are border roads. Figure 12 shows how 2 main roads split a map in 4 areas.

This map partitioning considerably reduces the searched area when a route needs to be determined between a source and a destination placed in different zones. First, a route is determined in the area of the source, from the source to the closest main road. The same is done in the destination’s area, between the destination and the closest main road. Last, a route is searched only using the main roads network between the points previously found. This approach reduces significantly the area being searched and also provides a route suitable for human drivers because it mainly contains major roads.

During the map partitioning process, some particular cases appeared:

- generally major roads surrounding one area form a grid for every node in the secondary roads. For routing inside an area, major roads are not absolutely necessary (Figure 13.A)

- in some cases major roads surrounding secondary roads form a grid in which not every node is connected to the others. In this case, for routing inside this area, main roads are required (Figure 13.B).

-not all maps are perfectly divided by major roads. Two different main roads can be connected and can form together just one longer road (Figure 13.C) or two secondary roads, each belonging to a different area can be connected. This is why after the process of assigning an area code is finished, a merging process takes place. This one looks for such abnormalities and merges such areas (Figure 13.D).

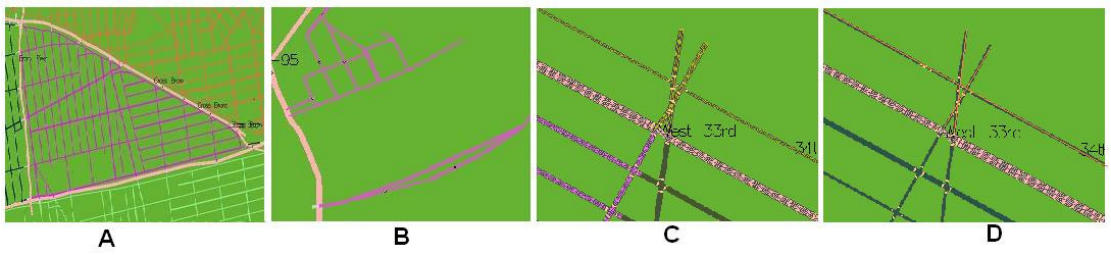


Figure 13. A-All secondary roads in a sub-network are connected;
B-Not all secondary roads in a sub-network are connected;
C- A split before merging
D-The same split after merging

After the splitting process ends, the results are saved in a file so that they can easily be loaded when needed.

Figure 14 presents the UML class diagram for the map construction and splitting.

The map contains a list of all roads. For each road details like road's name, type, lane number and allowed driving direction are kept. Each road has a list of consecutive points that form the road and a list of crosses.

For a road, crosses keep track of the points in which it intersects with other road, the indexes of those roads and the points on those roads where they intersect.

A point is described by its latitude, longitude and distance to the beginning of the road. For performance purposes, the concept of PeanoKey was introduced. A PeanoKey is in fact a number formed by interleaving the digits of the latitude with the digits of the longitude values. In this way a unique number is formed that describes a point and by keeping a sorted list of PeanoKeys searching for a point becomes much faster.

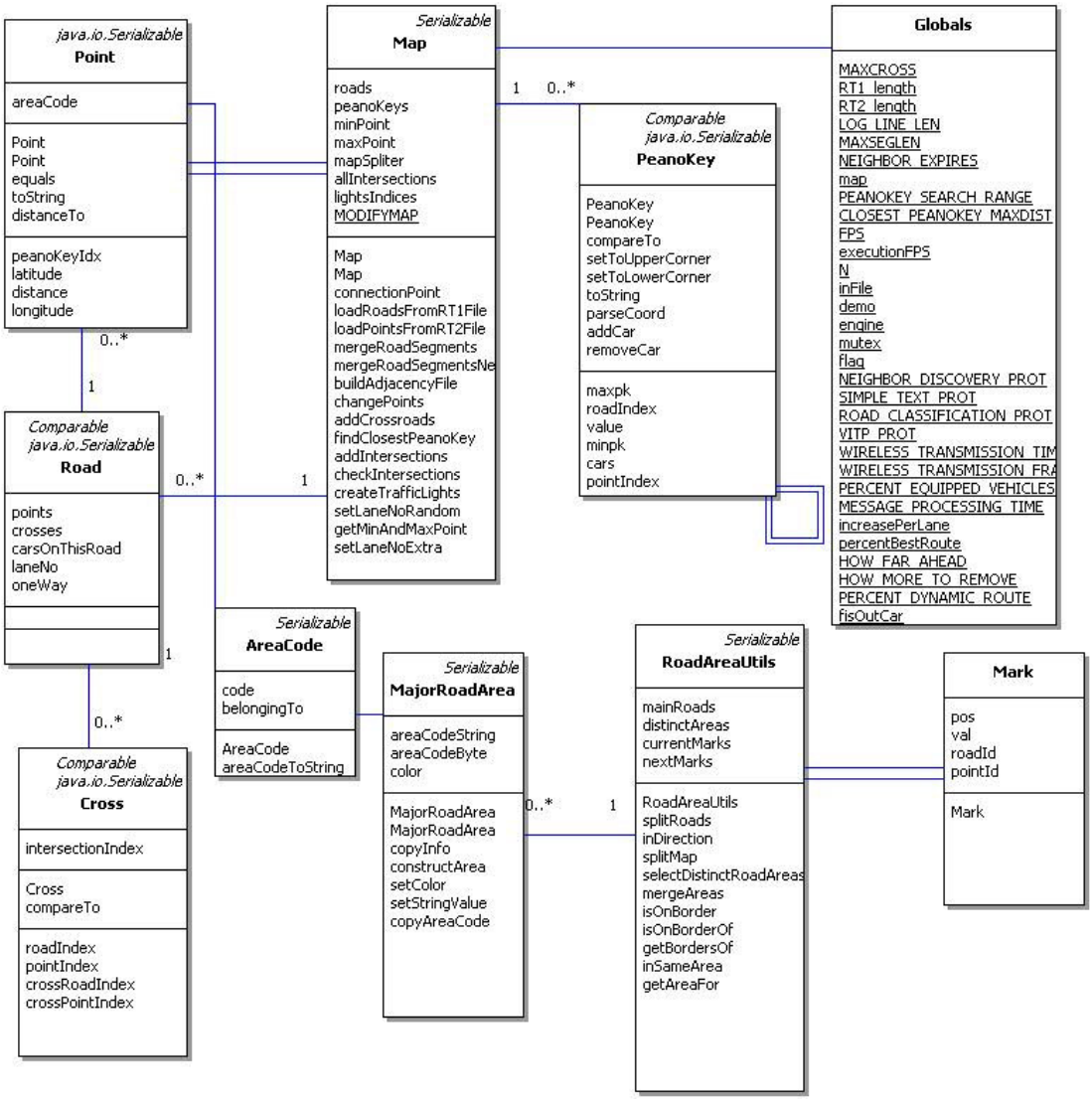


Figure 14. Map construction and splitting process

Before starting the splitting process, to each point an “AreaCode” object is assigned. The “AreaCode” has two components. The first is a list of bytes, one for each major road, that specifies the position of this point compared to that major road (left / right) and it is used in the first step of the splitting process when values are propagated from point to point. The other one is a “MajorRoadArea” object

that is used in the merging process. After each point's area code was set, a list of distinct "MajorRoadAreas" is formed, each point having a reference to the object corresponding to its area. This is extremely useful in the merging process, because when two areas have to be merged is not necessary to find all points in the area that is being deleted and change their values. It is enough to change the "MajorRoadArea" corresponding to the deleted area and all points will have the new information.

The "RoadAreaUtils" class is the one that manages the splitting process. It starts by making a list of all major roads and assigning each point an empty area code with a digit for every main road. Then, for every cross on a main road, it computes which point on the crossing road is on the left and which is on the right. It creates a list of "Mark" objects for all points found on secondary roads, next to main roads. Then, until no "Marks" will be added, it analyses the list of "Marks" objects and completes the corresponding point's code, adding a "Mark" object for every neighbor that was not yet visited. After all points are assigned an area code the merging process begins.

The static "Globals" class is the one that keeps a reference to the "Map" object so that it can be easily accessed from all other classes.

3.2. Best route computation

Roads form a directed and weighted network G = (N, A) with a set of nodes N and a set of arcs A. Nodes are generally road crosses but can also be just points on a road. The arcs are streets, more exactly road segments between two crosses. Each arc (i, j) from A has a weight given by an approximation of the travel time between i and j. The direction of the arc corresponds to the allowed driving direction of the corresponding road.

In order to find a route between a source and a destination, the algorithm first compares the area codes of the two points. If they belong to the same area, a

road between the two is searched. If they belong to different areas, the algorithm first searches for a route between the source and a main road, then a route between the destination and a main road and finally for a route on main roads between the points previously computed. The algorithm that is used for searching these routes is Dijkstra's shortest path algorithm with a special cost function.

The cost function used for computing the path is an approximation of the travel time. This approximation of the travel time between two points is considered to be the sum of the approximated values obtained for each segment of the route. A segment is considered to be the area between two points of the same road in which driving conditions don't change. Generally this is the area between two crosses. The value computed for each segment is based on the road's type, number of lanes and maximum speed allowed. The formula used for computing this value is:

$$\frac{distance}{maximum\_speed\_allowed} * \frac{[(3 - laneNumber) * delayPerLane] + 100}{100}$$

The *"delayPerLane"* factor is a way to add a delay that might appear on a road with fewer lanes compared with another with more lanes. A three lane primary road or highway is considered to be the best option. Fewer lanes will increase the travel time while more lanes will decrease it.

The statically computed route is kept by each car as a succession of route segments. A route segment is in fact a section of a road between two crosses. Using this list and an index to the segment on which the car is currently positioned, the car sends inquiries about the next few segments it will travel on.

The first type of inquiry was intended to gather the speed of cars traveling on the road segments ahead and maybe information about particular road conditions like ice or accidents. When this information returned to the car that initiated the query, the average speed would be computed and if it were under a specific limit or if a particular road condition were reported, the car would then

compute a detour. But this type of information proved not to be suitable for a city scenario due to the fact that in a city the cars that reply to an inquiry may be queuing at a traffic light and have "zero" speed. Reporting this speed would mislead the inquiring car to consider a jam on that road and avoid it, when in fact, it is not the case. This is why a new reply type was considered.

It was assumed that a car can approximate the time needed to pass over a road segment and that waiting just one red light period at the traffic light in the end of that segment is acceptable. This is why cars will try to find how long it took other cars to travel along the road segment of interest.

Each car will keep a record of the time when it started traveling on a new road segment. When a query is made, only cars at the end of the destination segment will reply and will give as information the time it took them to travel that segment.

When the reply reaches the initiator of the query, it performs an approximation of the time required for traveling on that road segment plus an approximation of the time required to wait for a red light at the traffic light in the end of the segment. If the average value reported is bigger than that approximation, it computes a detour.

The formula used for computing the maximum time allowed for traveling a road segment is:

$$\frac{segment'sLength(km)}{\frac{2}{3} * maximumSpeedAllowedOnThatRoad} + (nrSegmentsEnteringTheCross - 1) * 35 \quad (s)$$

The first fraction is an approximation of the time required to travel a road segment depending on the maximum speed allowed on that type of road and on the length of that segment. Because in a city scenario cars don't generally travel at a

constant speed, often breaking and then accelerating, assuming a car travels constantly at the maximum speed allowed is not realistic. This is why an estimation of 2/3 of the maximum speed allowed was used.

The duration of a red light is approximated by the maximum amount of time required for all other segments entering the intersection to have a green light. Depending on the other roads' type, some might have a shorter or greater period of green light than others. The average between the different green light periods was used, which means 35 seconds.

For a highway scenario this formula had to be modified because on a highway no traffic lights exist. This is why only the approximation of the time required to travel on that road segment with a speed of 2/3 from the maximum speed allowed was used.

3.3. New Route Computation

When a car receives a reply that signals a very long travel time on a street segment, a detour from the original route will be computed. The new route will have to go around the road segment about which the car now knows it is jammed.

Assuming that the initial route was an optimal one, there is no sense in computing the whole route towards the destination again. This is why a new route will just try to go around the problem segment and to reach the route previously computed in an area after the avoided segment. This detour will be computed using Dijkstra's shortest path algorithm with distance as a cost function.

Figure 15 presents the UML class diagram for the dynamic route computation process.

A real car is described by the "Car Instance" Java object. Each car is uniquely identified by an id and it keeps track of its position by means of an index of the current road and current point. The route followed by a car is described by a

list of route segments and an index pointing to the current route segment. The car has a driver personality associated that is used by the mobility model. For the routing protocol, cars keep a record of the time when they entered the latest road segment and have a random moment during the query period when they send inquiries.

The "RouteComputingUtils" class contains all the functions used in computing static routes. The basic algorithm used was Dijkstra's shortest path algorithm with variations for the cost function and the area searched.

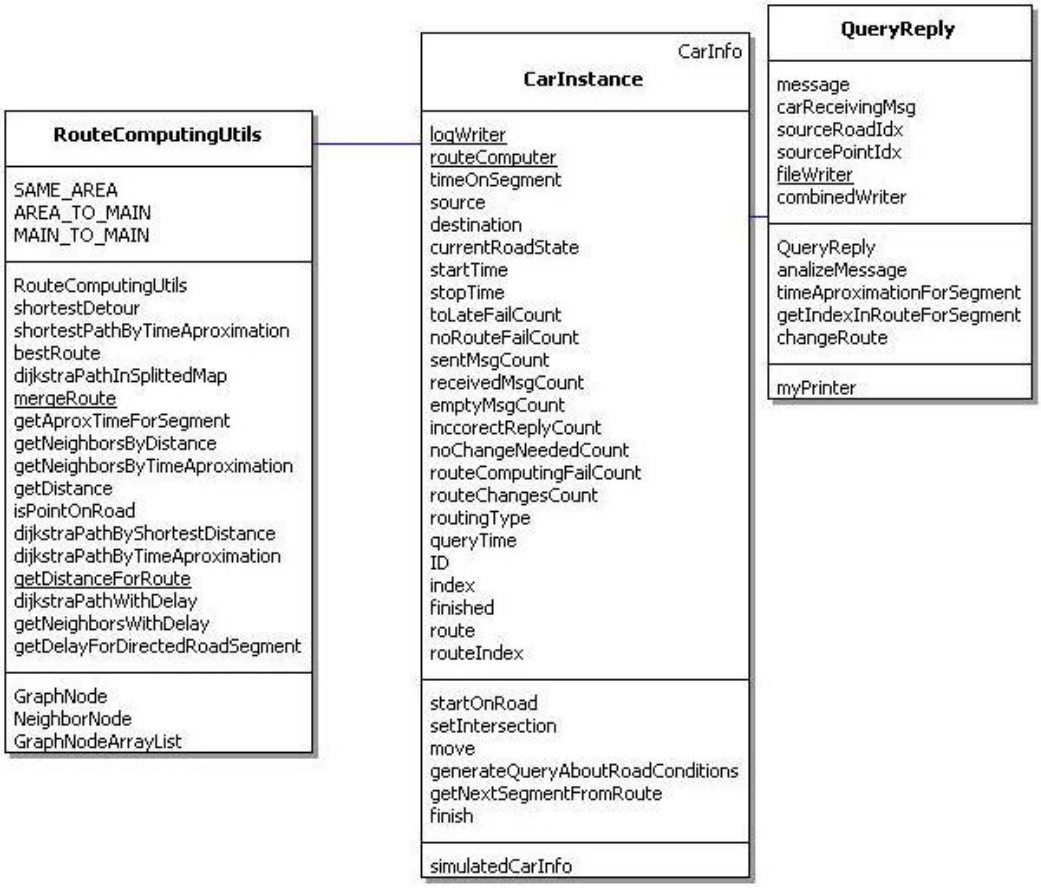


Figure 15. UML class diagram for the route computing process

When a car receives a reply, a “QueryResponse” object is generated. This object analyzes the received message and validates it. If the message is valid, an approximation of the maximum time allowed for a car to travel the segment in question is done. If the average time reported by other cars is bigger than the computed value, a detour is computed. If a detour is found, the car’s route is then changed.

4. Dynamic Route Computation Using a Hybrid System

Using queries to gather information about road conditions is a good method to dynamically route a car around problematic areas when a car already has a route that is considered optimal. As the results presented later in this paper will show, this method gives good results when only a percent of all cars use this type of routing. This is due to the fact that the information gathered covers just a small area of interest, not taking into consideration the big picture. When a large percentage of vehicles traveling on a road determine in the same time a condition that makes them change their route, they all move on another road, many times creating a jam on the new road.

This is why, in order to create a dynamical routing protocol that gives even better results, more information about road conditions is needed. If all cars start to inquire at the same time about all streets around them, trying to gather as much information as possible before making a decision, a lot of communication is produced.

The solution presented in this chapter uses a hybrid system formed by both infrastructure nodes and vehicles.

This solution assumes that in each intersection a fixed node can be installed, capable of communicating with cars approaching the intersection through wireless communication and all fixed nodes are connected together with a separate high speed network.

Big cities around the world have already installed in intersections adaptive traffic lights that receive information from loop detectors or cameras that send information to a control center where decisions are made about how to control traffic. In these conditions installing a wireless communication device with processing power and assuring a connection between these points is not a very burdensome operation.

When a car starts on a new road segment, it sends a message to the fixed node ahead, communicating its destination. Having access to a database containing information about traffic conditions on all the roads, the fixed node can compute the shortest path towards that destination and reply to the car that interrogated it with the next segment from the found route. This process repeats itself for every road segment because the vehicle flow is highly dynamic and conditions change very rapidly, so it is better to interrogate at every step the fixed node and always get the best route at that particular time.

Initially it was intended for the fixed nodes to keep information about how long it took for the last cars traveling a segment to pass on that segment. Each car leaving a road segment would communicate to the fixed node the time it took it to travel that segment and the average of the times reported by the last ten cars leaving that segment would be communicated to other fixed nodes so that they can used it in the routing process.

This method did not produce good results due to the fact that the cars that report their values can be the first in a long queue. Because they can be the first that entered that segment, the times reported by them will be small. This will trick the fixed nodes to route even more cars on that segment, only contributing to the increase of the queue. Until the cars in the middle or in the end of the queue reach the intersection and report big waiting times, many more cars would have been routed to that segment creating an even bigger jam.

Due to this fact, it became clear that the number of cars already on a segment has to be taken into consideration when computing an approximation of the travel time for a particular road segment. Their presence induces a delay that can be computed based on how fast the intersection at the end of the segment can let them pass on to another segment.

For every road segment entering an intersection the following values were computed:

-the ***capacity*** of that segment is the number of cars entering the intersection from that segment that can pass the intersection in an hour; knowing that the mobility model allows 0.5 cars per lane to pass the intersection per second and knowing exactly the traffic light's cycle length and the green period for that segment, the number of cars that can pass the intersection per hour for a road segment is:

$$capacity = 0{,}5 * nrLanes * \frac{greenTimeForTheSegment(s)}{TrafficLightCycleLength(s)} * 3600$$

-the ***approximateTravelTime*** required to pass a road segment if no other cars would be on that segment:

$$\frac{segment'sLength(km)}{\frac{2}{3} * maximumSpeedAllowed} + (TrafficLightCycleLength - GreenTimeForTheSegment)$$

At regular time intervals, a fixed node counts the cars on the segments entering its corresponding intersection and computes for each segment the time in which a car entering that moment would have to wait to pass the segment and enter the intersection:

$$approximateTravelTime + \frac{nrCars}{capacity} * 3600 \quad \text{(s)}$$

This value is then communicated to all other fixed nodes so they can use it in the process of finding a route.

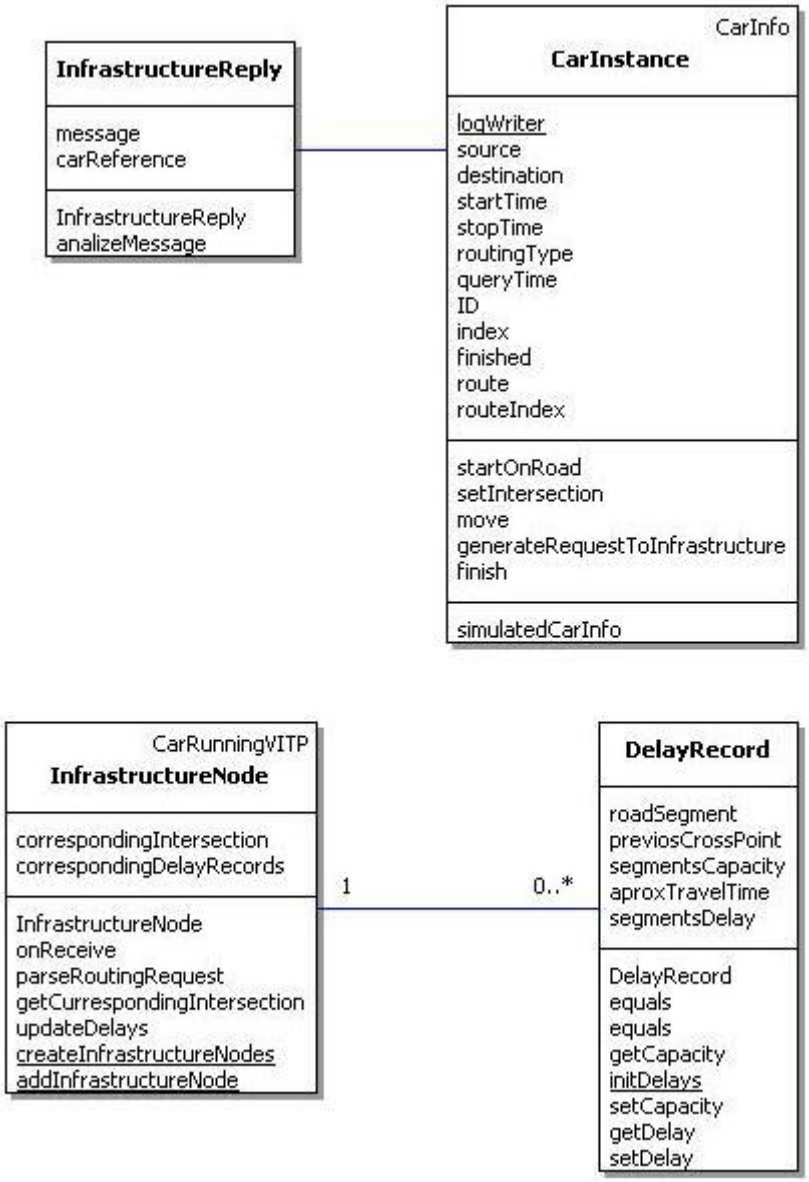


Figure 16. UML class diagram for the hybrid route computing mechanism

The "DelayRecord" class is the one that keeps the information regarding the current traffic conditions associated with a road segment. For each segment the following data is kept: a description of that segment's position, that segment's capacity and the approximated travel time when no cars are on that segment. The

segment's delay is recomputed periodically as the sum of the approximated travel time and the time required for all the cars already on that segment to pass the intersection.

A fixed node is described by an "InfrastructrureNode" Java object. Each node contains a reference to the corresponding intersection and references to the delay records corresponding to the road segments entering the intersection. It also contains methods for answering a routing request, for computing the shortest route based on the delay information for all segments and for updating the delay information for the road segments entering the intersection.

Cars are described by the "Car Instance" class. They have a source, a destination and a list of segments they traveled on. The last in this list is generally the current segment or, when an answer has been received, the next segment. Because a car doesn't know from the start on where it will travel, each time a new segment is added to its route some parameters corresponding to the mobility model have to be computed in order to set the way in with the car will transit from one segment to another.

When a car enters a new road segment it will generate a routing request to the fixed node at the end of the street periodically, until an answer is received. After an answer is received, no more messages will be generated until the car moves to another segment.

When an answer is received, a "RoutingResponse" object is generated that has a reference to the car object and the message. This object contains the methods that parse the reply, check it for correctness and if found to correspond, the segment indicated in the response is added to the car's route.

5. Results and Discussions

5.1. Testing Conditions

All tests have been performed on computers with a 2.8GHz processor, with 1,24GB of RAM, running WindowsXp.

For the first algorithm two scenarios were created: a city center scenario and a highway scenario. The percentage of cars using a simple shortest path algorithm to compute their route and the percentage of how many use the algorithm described at 4.2 were varied. Another factor that was studied was the percentage of cars using the greedy mechanism compared to the ones that just followed their route. Last, the distance to which an interrogation should be send was varied.

The first scenario was an urban city scenario using a small part of Manhattan's map. This scenario had 17 Km of roads, 32 entries and 32 exits were defined. On each entry a flow/lane/hour of 200 vehicles was induced for 10 minutes. Then all cars were left to reach their destination. For cars entering the map from one point, destinations were equally assigned to all exits that were more than 250m away (Figure 17A).

In this scenario the wireless range of the vehicles was set at 200m and each communicating vehicle generated a new query at a 30 second period. This proved to be a sufficient query period for the city scenario due to the following factors:

-all segments forming a route were very short (generally hundreds of meters between two road crosses) so the answers would travel back very quickly.

-all intersections had traffic lights, and due to the high density of cars, a car would generally catch a red period at a cross. The red light period was around of 35 seconds, which is more than enough time to get even more than one reply.

The second scenario was a highway scenario using a part of New Jersey Turnpike's map. This scenario had 87 Km of roads, 24 entries and 24 exits. On each entry a flow/lane/hour of 400 vehicles was induced for 10 minutes. Then all cars were left to reach their destination. For cars entering the map from one point, destinations were equally assigned to all exits that were more than 1Km away (Figure 17 B).

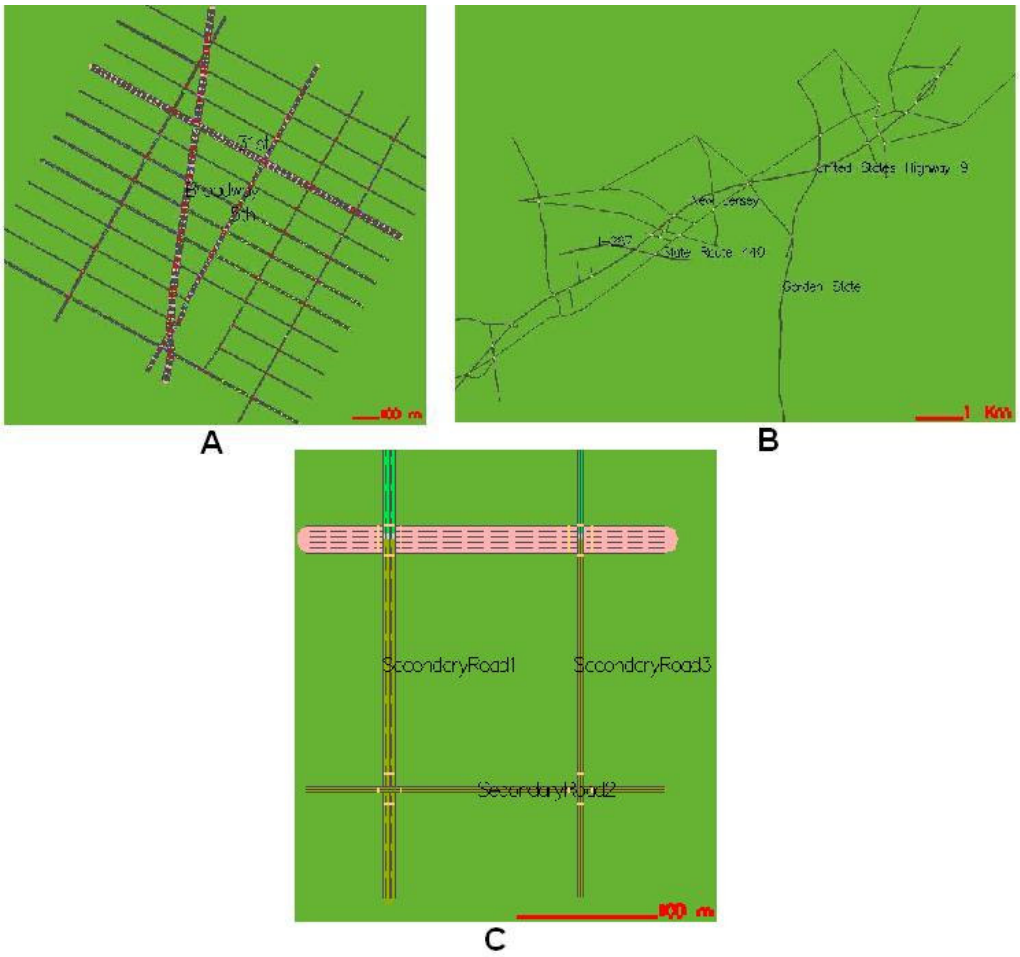


Figure 17. Test Scenarios

After completing their trip, each car added an entry to a log, containing its id, its origin, its destination, the total travel time, the distance traveled, and the routing type used. After all cars finished, this log was analyzed and the average time and distance were computed.

In this scenario the wireless range of the vehicles was set at 500m. Due to the high speeds cars travel on a highway and the lack of traffic lights, the period of 30 seconds for sending a query was too big. The retransmitting period was set to 10 seconds for this scenario and every time a car entered a new segment, a query would be automatically generated.

For the hybrid mechanism two scenarios were used: first a simple scenario made of 4 roads (Figure 17 C) and second the city scenario presented above.

5.2. Results Obtained

The map splitting process was tested on different maps with different sizes. As main roads two types of roads were considered: interstate/state highways or both highways and primary roads. The times obtained for running the splitting algorithm itself are presented in table 1.

Map Name	Total length (Km)	Number of Roads	Number of major roads	Split process duration (ms)
DowntownNY	**17**	**23**	**3**	91
NJTurnpike	**196**	**50**	**4**	811
			5	861
NYState	**1402**	**2373**	**60**	465850
			64	484147
NJState	**4670**	**10400**	**20**	89639
			41	762606

Table 1. The duration of the map splitting algorithm

From these values it can be deduced that the time required for the splitting process is directly proportional to the total length of the map roads but the number of major roads is the factor that influences the most the duration of this process.

Table 2 presents the average times required for two algorithms to find a route between two points. Four distance intervals were measured. For each distance interval, 100 pairs (source-destination) of points were generated. Each algorithm was applied for these values. Table 3 presents the total distance produced by each algorithm.

The first algorithm (Dijkstra) computes the shortest path using the approximated travel time as a cost. The second, presented in 3.2 (will be referred from now on as "best route algorithm"), computes the shortest path from the source and from the destination to a main road and then the shortest path between these two points on main roads. It also uses the approximated time as a cost function.

	0 to 10 Km	10 to 20 Km	20 to 40 Km	40 to 100 Km
	Average time (**ms)**	Average time (**ms)**	Average time (**ms)**	Average time (**ms)**
Dijkstra Shortest Path Algorithm	378	1139	1864	3279
One route using mainly primary roads	267	566	817	1342

Table 2. Average times required for two algorithms to find a route between two points

These values prove that the algorithm presented in 4.2 is much faster than Dijkstra's shortest path algorithm. Because for Dijkstra's algorithm the cost function was an estimation of the travel time, estimation that is powerfully influenced by the road type, even this algorithm gives a solution that uses primarily major roads.

	0 to 10 Km	10 to 20 Km	20 to 40 Km	40 to 100 Km
	Total distance **(km)**	Total distance **(km)**	Total distance **(km)**	Total distance **(km)**
Dijkstra Shortest Path Algorithm	566.1	1510.6	2875.9	4280.5
One route using mainly primary roads	622.1	1479.9	2889.3	4263.1

Table3. Total distances produced by the two algorithms when finding a route between two points.

The other one, using the same cost function but having a smaller search area, is much faster and produces a solution that is very close to the one obtained from the first algorithm. This can be concluded by analyzing the total distance produced by both algorithms.

Chart 1 presents the average travel times obtained by cars in the urban scenario when traveling on a static route computed with one of the two algorithms (shortest path and the best route algorithm). Multiple measures were made, varying the proportion in which each algorithm is used for the computation of the initial static route.

From these values it can be observed that using only one type of static route computation produces the longest journeys. This is because all cars go on the same streets, creating a very high density of cars in those areas. Between the two algorithms, when used just one or the other, the shortest path algorithm produces longer travel times than the best route algorithm. This can be explained by the fact that the shortest path algorithm uses mainly small streets, while the best route algorithm uses predominantly major roads that have a bigger capacity, have longer green periods in intersections and even greater speeds allowed, although this last criterion does not apply in cases of jams.

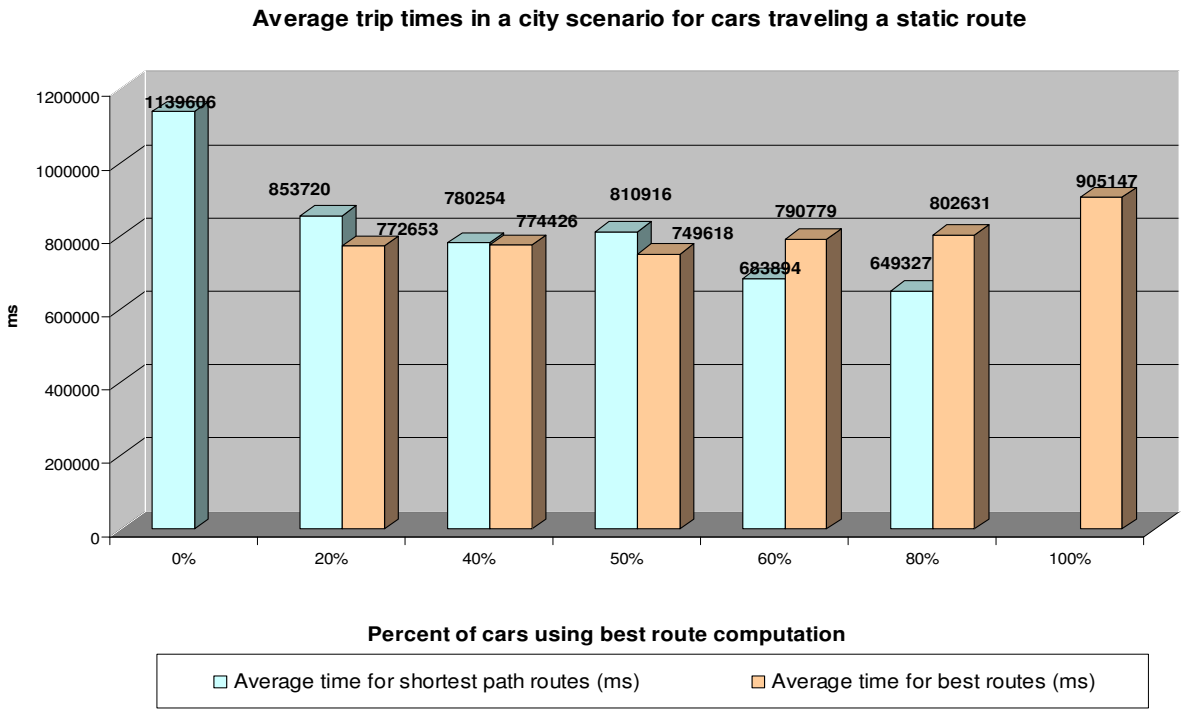
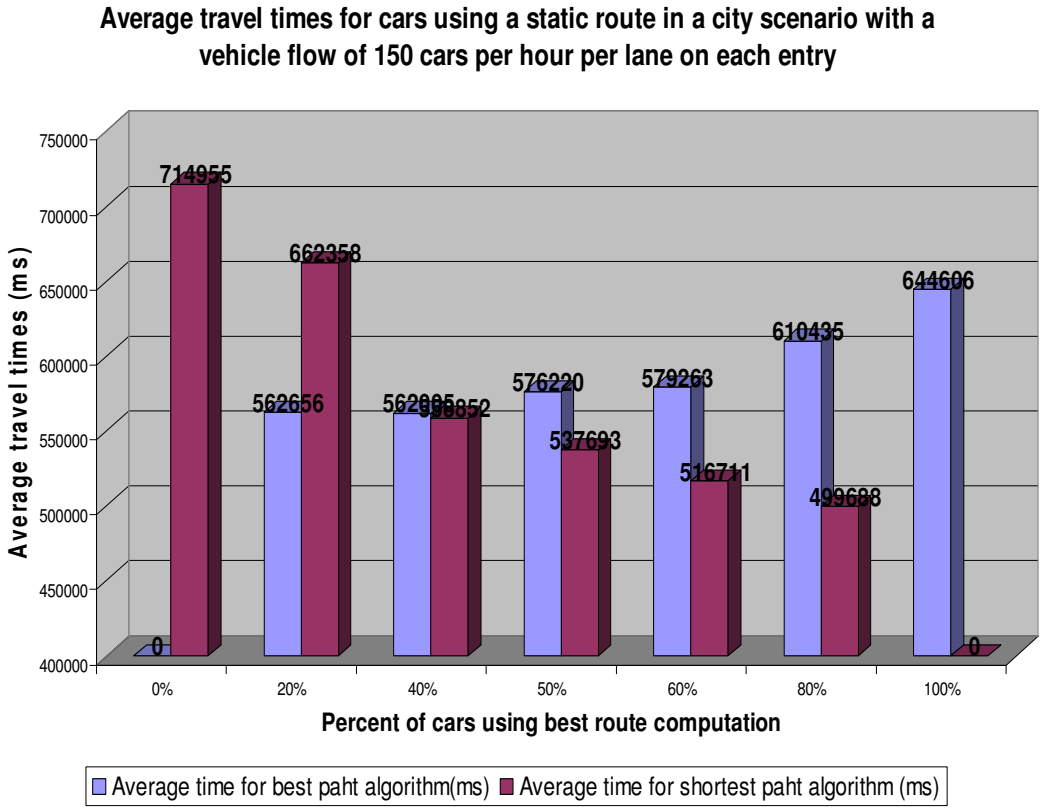


Chart1. Average journey times (ms) in city scenario created by injecting 10 minutes a flow of 200 vehicles/lane/second on each of the 32 entries.

The best average travel times are obtained when a combination of the two algorithms is used. The data in the chart above presents as the best combination a 60% usage of the best route algorithm and a 40% percent usage of the shortest path algorithm. Again this can be explained by the fact that the roads used by the first have greater capacity.

For the same scenario, the vehicle flow was reduced to 150 and 100 vehicles/hour/lane for each entry and the same measurements were done. The results are presented in Chart 2 and Chart 3.



Char2. Average journey times (ms) in city scenario created by injecting 10 minutes a flow of 150 vehicles/lane/second on each of the 32 entries.

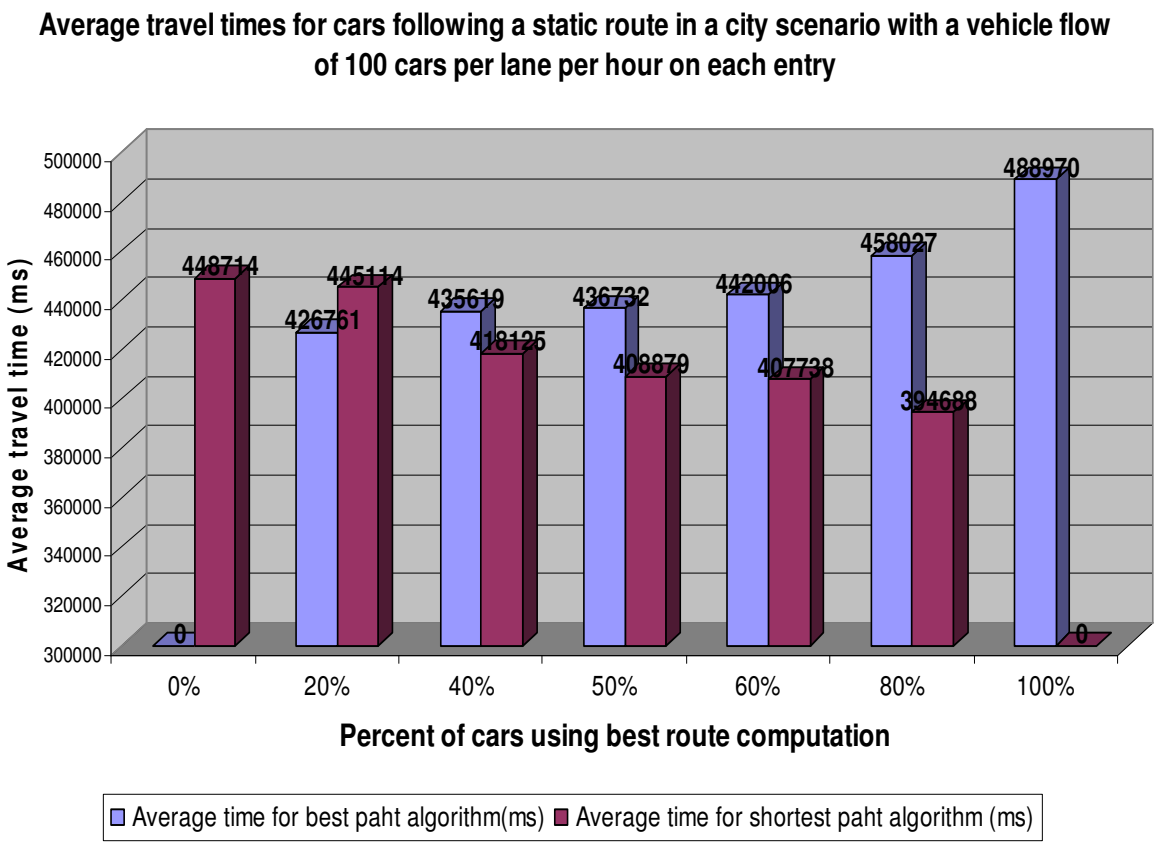


Chart3. Average journey times (ms) in city scenario created by injecting 10 minutes a flow of 100 vehicles/lane/second on each of the 32 entries.

The results obtained from these three simulations prove that the average travel time increases with the increase of vehicle flow. For the scenario with the smallest vehicle flow, the average travel times obtained by using only a static route computed with the shortest path algorithm is smaller than the average travel time obtained when using only a static route computed with the best route algorithm. This can be explained by the fact that in a scenario where traffic flow is light, cars travel easily on all types of roads and the ones that have a shorter distance to travel will be the ones that will get there faster.

This city scenario was used for testing the hybrid routing system. The average travel times obtained for different vehicle flows are presented in table 4.

Vehicle flow per entry (vehicle/lane/hour)	**100**	**150**	**200**
Average travel time for hybrid road planning system	399718	539130	725827

Table 4. Average travel times for cars in the city scenario using the hybrid routing system.

Compared to the values obtained in the cases when only a static route was used, these values are just slightly bigger than the smallest values obtained in each simulation. The smallest values belong to the cars that follow a shortest path in a scenario where the majority of the cars follow a best route path.

Because the values produced by the hybrid routing system apply to all traffic participants, not to a small fraction of traffic participants, this system can be considered better than static routing.

Compared to the times when only a type of static routing is used, the hybrid routing systems produces average travel times that are 20-30% percent smaller.

Next, for the greedy approach, the city scenario with a flow of 200 vehicles/lane/hour was used. The percent of cars that uses the dynamic routing mechanism was varied and the number of road segments ahead was also varied.

Table 5 presents the average travel times obtained by cars in the city scenario when 60% of them compute their initial route with the best route algorithm and 33% from all cars communicate in order to determine road conditions and find a detour if needed.

The first thing that can be observed from these values is that the best results are given by interrogating road conditions just one segment away. This can be explained by the fact that when avoiding an area further away, the detour might prove to be longer and may include narrow streets and more intersections. In many cases until the cars reach that segment the jam might have been softened. When interrogating just one segment away, imminent jams are avoided and resulted detours shorter, containing less traffic lights.

Number of road segments ahead to be interrogated by the vehicles which use queries **Road computation Process**	**1**	**2**	**3**
Average travel time for cars traveling a static shortest path route (ms)	619256	629864	644188
Average travel time for cars traveling a shortest path route and communicating (ms)	567024	584314	655123
Average travel time for cars traveling a static best path route (ms)	697966	713101	745059
Average travel time for cars traveling a best path route and communicating (ms)	667185	672849	706163

Table 5. Average travel times for cars in the city scenario where 60% compute their initial route with the best route algorithm and 33 % of all cars communicate in order to determine road conditions.

Another thing that can be observed is the fact that even the cars following the static route are influenced by route modifications of other cars. The further the area that has to be avoided, the more the cars leaving major roads stay on secondary roads, increasing the density of cars on these roads and producing longer travel times.

Table 6 presents the average travel times for cars in the city scenario where 60% compute their initial route with the best route algorithm, inquiring about 1 road segment away on their route. The percent of cars that use communication in order to determine road information is varied.

Percent of cars communicating in order to determine road information **Road computation Process**	**10%**	**33%**	**60%**	**80%**
Average travel time for cars traveling a static shortest path route (ms)	662462	619256	672710	726023
Average travel time for cars traveling a shortest path route and communicating (ms)	564055	567024	682189	696656
Average travel time for cars traveling a static best path route (ms)	754932	697966	761089	757512
Average travel time for cars traveling a best path route and communicating (ms)	642014	667185	745837	804730

Table 6. Average travel times for cars in the city scenario where 60% compute their initial route with the best route algorithm, inquiring about 1 road segment away on their route.

The best overall values are obtained when only one third of all cars use the dynamic routing protocol. When only a few (10%) use this protocol, their values are also small, but the average times of the remaining cars are still big. The greater the percent of cars using this protocol the greater the average times become.

An interesting thing that can be observed is that the travel times for the cars using just a static best route algorithm become smaller than the times of the cars using the dynamic routing protocol when the last are 80%. This may be due to the fact that when a big percent of cars determine bad road conditions and all decide in the same time to change their route, they all move in the same time to secondary roads, creating even bigger jams on this smaller road and freeing the bigger road. This aspect is also reflected in the increase in the times obtained by cars using a static shortest path route and traveling mainly on smaller roads.

Table 7 presents the average travel times for cars in the city scenario where 33% of the cars communicate in order to determine road information, inquiring about 1 road segment away on their route. The percent of cars using an initial static route computed with the best route algorithm was varied.

When just 40% used best route computation, the average travel times obtained by cars using dynamic routing were not very different from the ones obtained by the cars using a static route. This may be due to the fact that the percent of cars using primary road, which are preferred by the best route algorithm, was small enough not to produce jams.

When all vehicles used best route algorithm for computing the static route, the average times of the ones that did not use the dynamic routing mechanism decreased just slightly compared to the situation when none used the routing mechanism. The average travel times of the ones that used the dynamic routing mechanism were not significantly smaller either.

Percent of cars using an initial route computed with the best route algorithm **Road computation Process**	**40%**	**60%**	**100%**
Average travel time for cars traveling a static shortest path route (ms)	694365	619256	NA
Average travel time for cars traveling a shortest path route and communicating (ms)	666298	567024	NA
Average travel time for cars traveling a static best path route (ms)	761837	697966	882076
Average travel time for cars traveling a best path route and communicating (ms)	762145	667185	842379

Table 7. Average travel times for cars in the city scenario where33 % of all cars communicate in order to determine road conditions, inquiring about 1 road segment in front.

The next studied environment was a highway scenario. Out of New Jersey's state map a small area around NJ Turnpike highway was isolated, with a total of 82 Km of roads. The tests performed on this map have proven that the

highway had enough capacity to accommodate all generated cars without creating jams, so no concluding results were obtained just from the variation of the parameters described in the previous experiment.

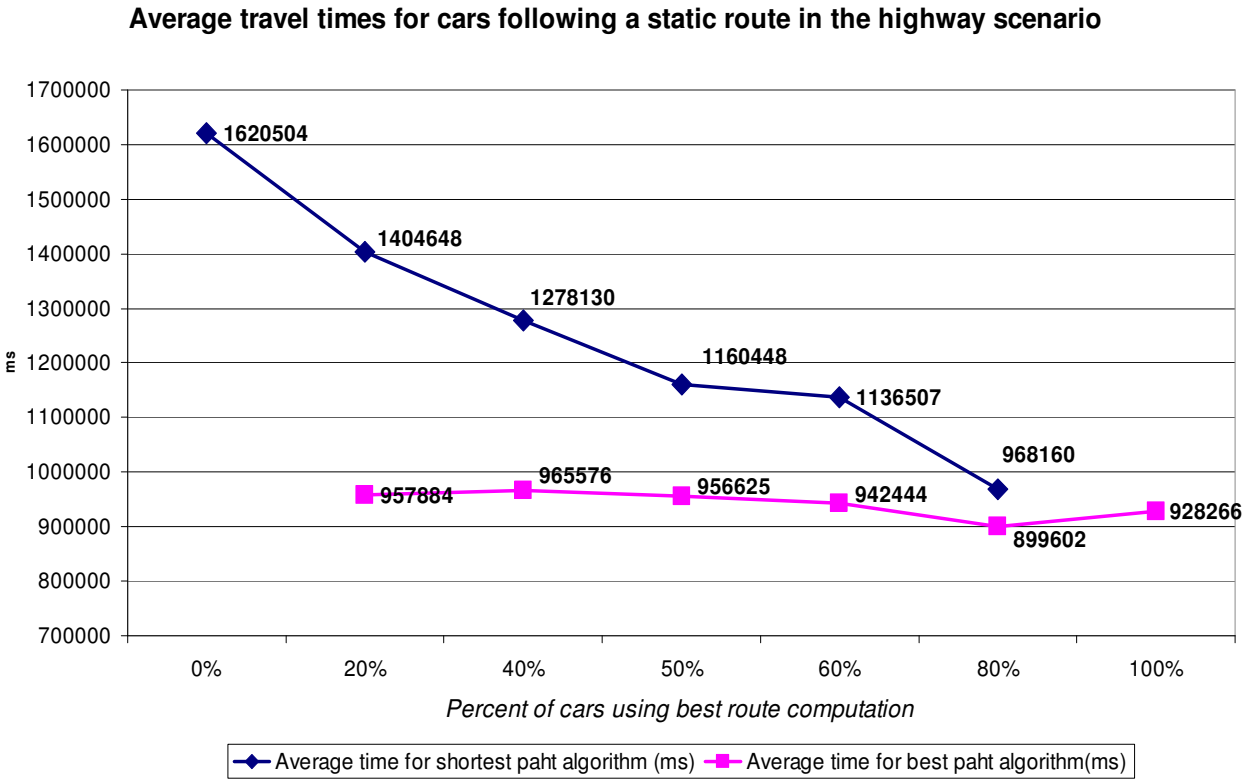


Chart4. Average journey times (ms) in a highway scenario created by injecting for 10 minutes a flow of 400 vehicles/lane/hour on each of the 24 entries.

Chart 4 presents the average travel times obtained by cars in the highway scenario when traveling on a static route computed with one of the two algorithms (shortest path and the best route algorithm). Multiple measurements were made,

varying the proportion in which each algorithm is used for the computation of the initial static route.

For the cars using the shortest path algorithm, the pattern from the city scenario was preserved. The average travel times decreased along with the decrease of the number of cars using this algorithm. This was due to the fact that secondary roads which were mainly used by this algorithm had the same type of limitations the secondary roads in city scenario (traffic lights, smaller capacity).

For the cars using the best route algorithm, no significant variations were recorded. I attribute this to the fact that the highway, which was preferred by this algorithm, had enough capacity to accommodate the traffic and jams were not created.

In order to produce more interesting results, for this scenario a number of dummy cars were created that stopped in the middle of the highway, in order to induce congestions.

During this scenario the following observations were made:

- communicating cars detected the jam and computed an avoidance route.

- the detour involved leaving the highway and using secondary roads for the avoidance route.

-when these cars tried to enter back on the highway, jams were created at the entrance point.

The mobility model used by the simulator describes intersections by means of traffic lights or priority signs. No information is included about special access lanes to the highway. Access to the highway could be made by means of traffic lights or priority signs.

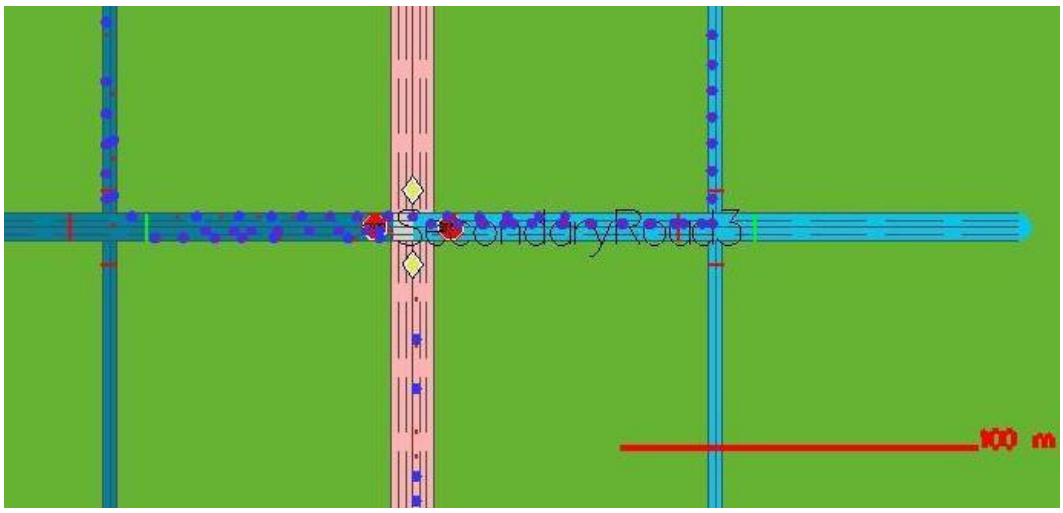


Figure 18. Highway jam produced due to the fact that cars on secondary roads wait for the cars on the highway to pass, giving them priority

Simulating traffic lights on the highway at each crossroad with a secondary road would mean reducing the scenario to the city scenario. Having only priority signs in these crosses and allowing cars on the highway to travel freely creates jams at the access points on the highway (Figure 18). The jams are produced because the entering cars see the cars traveling on the highway and wait for them to pass before entering the highway. In real life, cars use the special access lanes, accelerate and then easily change lanes.

This is the main reason for which this scenario could not produce valid results.

6. Conclusions

I have developed two applications that aim at reducing the vehicles' total travel time towards a destination by dynamically computing the best route, using real time information about congestions and road conditions. Both solutions assume that cars are equipped with short-range wireless communication devices and have processing power.

The first application initially computes a static route and then, using car-to-car communication, determines traffic conditions on the roads it will travel. When unfavorable traffic conditions are detected, the car computes a detour avoiding the area with problems. The solution was evaluated on two scenarios: a city scenario and a highway scenario. For the city scenario this solution proved to produce good result for conditions in which the static route is computed using a combination of algorithms and when only a fraction (around 30%) of all cars used it. For the highways scenario, due to the limitations of the mobility model, no complete results could be produced.

The second application presented was a hybrid system composed both of infrastructure nodes and vehicles. The fixed nodes are placed in all intersections, are connected and share a database with information about traffic conditions. Cars approaching an intersection request routing information from the fixed node, communicating to it their destination. Based on the information it has about jams and delays, the fixed node computes the best route and replies to the vehicle with the next road segment it should go on. This application was tested on a city scenario in which the vehicle flow was varied. This method has produced good results for all traffic participants.

Although some average travel times obtained by the first application are smaller than those produced by the second, it can be concluded that the hybrid system is better because the times produced by it apply to all traffic participants while the times produced by the first apply to fractions of participants and only in specific conditions.

.

7. References

[1] Intelligent Transportation Systems - U.S. Department of Transportation . http://www.its.dot.gov/index.htm

[2] Bin Liu- "Using Knowledge to Isolate Search in Route Finding". *International Joint Conference on Artificial Intelligence*, pp.119-125, 1995

[3] Diaconescu Raluca, Gorgorin Cristian, Gradinescu Victor – "Integrated Vehicular Traffic Simulator". *Student's Scientific Convention - Politechnica University*, May 2006.

[4] P. Bovy and E. Stern, Route Choice- "Wayfinding in Transport Networks". Studies in Industial Organization. Kluwer Academic Publisher, 1990.

[5] P. Shankar, M. T. Alam, S. Musharoff, N. Ravi, C. V. Prados, V. Gradinescu, R. Diaconescu, C. Gorgorin and L. Iftode – *" Outdoor Experience with the TrafficView Application*", 2006

[6] Bin Liu – "Intelligent Route Finding: Combining Knowledge Cases and An Efficient Search Algorithm". In *Proc. of the 12th European Conference on Artificial Intelligence*, pages 380-384, 1996.

[7] G. Eggenkamp, L.J.M. Rothkrantz – "Intelligent dynamic route planning". *KBS & TRAIL Workshop*, June 2001

[8] M.Dikaiakos, S.Iqbal, T.Nadeem, L.Iftode – "VITP: An Information Transfer Protocol for Vehicular Computing". In *Proc. of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET)*, September 2005.

[9] The Network Simulator - ns-2 - http://www.isi.edu/nsnam/ns

[10] Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator (JiST / SWANS)- http://jist.ece.cornell.edu/index.html

[11] VISSIM - http://www.english.ptv.de/cgi-bin/traffic/traf_vissim.pl

[12] Traffic Software Integrated System - http://mctrans.ce.ufl.edu/featured/TSIS/

[14] Street Random Waypoint - vehicular mobility model for network simulations (STRAW) http://www.aqualab.cs.northwestern.edu/projects/STRAW/index.php

[15] US. Census Bureau – Tiger Map Service http://www.census.gov/geo/www/tiger/tigermap.html

[16] D.R.Choffnes and F.E. Bustamante - An Integrated Mobility and Traffic Model for Vehicular Wireless Networks. In Proc. of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET), September 2005.